DATA MINING FOR FEATURE VECTOR NETWORKS

by

Flavia Moser
M.Sc. Ludwig-Maximilians-Universität München, 2006

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Fall 2009

APPROVAL

Name: Flavia Moser

Degree: Doctor of Philosophy

Title of Thesis: Data Mining for Feature Vector Networks

Examining Committee: Dr. Diana Cukierman
Chair

Dr. Martin Ester, Senior Supervisor

Dr. Derek Bingham, Senior Supervisor

Dr. Oliver Schulte, Supervisor

Dr. Jian Pei, Internal Examiner

Dr. Hugh Chipman, Acadia University, External Examiner

Date Approved: September 17, 2009

Declaration of
Partial Copyright Licence

Abstract

Network data arise in various contexts. Examples include online friendship networks, email exchange networks and genetic interaction networks. In this dissertation, we are particularly interested in feature vector networks (FVNs). FVNs are networks wherein the nodes contain additional information in the form of feature vectors. In the case of social networks, the age or economic status of a person might be part of the feature vector. We investigate three research problems related to FVNs: clustering of FVNs, pattern mining in FVNs and simulation model for FVNs.

Clustering of nodes in FVNs is relevant, for example, in advertising on online social networks, where feature vectors could reflect recent purchases of users. The number of clusters in these networks is often unknown. We introduce a new approach for simultaneously identifying the number of clusters and the clusters themselves. Our objective function takes into account not only the network structure but also the feature vectors. The outcome of this partitioning clustering algorithm shows better results than current state-of-the-art methods.

Our pattern mining approach finds the set of all cohesive patterns. A cohesive pattern is defined as a connected, dense subgraph with similar feature vectors in a sufficiently large subspace. This definition is based on the needs of small communities in social networks and also on modules in Protein-Protein interaction networks. Our dense graph mining algorithm is the first to guarantee a complete result for density thresholds above $\frac{1}{3}$. Additional constraints on feature vectors and connectivity reduce the number of patterns and add additional meaning.

The lack of both publicly available FVNs and FVN simulation models motivates us to create a framework for simulating FVNs. We present a framework which is based on the previously introduced Latent Socio-Spatial Process (LSSP) model and on our extension, the n^*LSSP model. The n^*LSSP model uses two dampening functions in order to deal with effects related to the network size. Both models can be used to predict links in FVNs, and the model parameters are estimated using Markov Chain Monte Carlo.

Keywords: Social network analysis, statistical network analysis, graph data, clustering of network data.

To my family

"A ship in port is safe, but that's not what ships are built for."

Grace Hopper

Acknowledgement

During the work on this dissertation and my stay at SFU, I had the great opportunity to collaborate with many great people. First, I would like to thank my supervisory committee. I thank Dr. Martin Ester who supported me through all these years. I am greatly indebted to Dr. Randy Sitter who introduced me to the Statistics aspect of social networks and who was a great inspiration. I was very saddened and shocked when I heard the news of his loss. Thanks to Dr. Derek Bingham for his support and for teaching me to climb higher. My gratitude also goes to Dr. Oliver Schulte with whom I spent so many hours discussing various multi-relational Data Mining approaches. I appreciate the support of Dr. Jian Pei and Dr. Hugh Chipman for kindly agreeing to be my internal/external examiners and providing me with much valuable feedback. Finally, I would like to thank Dr. Crystal Linkletter for her support.

In the last few years, I had the chance to travel to many conferences, workshops and events. I am very grateful for this opportunity and for the financial support provided to me by various grant holders and grants. In particular, I would like to thank Dr. AC Surendran for my exciting internship in Seattle.

I also worked with many fellow students in the Computing Science, Statistics and other departments here at SFU. I am very thankful for their support and for sharing their knowledge with me. In particular, I would like to thank my close collaborators: Recep Colak, Richard Frank, Rong Ge, Fereydoun Hormozdiari, and Mona Vajihollahi. I am also lucky to have had very supportive labmates. Especially, I would like to thank my labmates for the past two years for their support and helping me survive the completion of my thesis: Jervyn Ang, Joslin Goh (there is nothing better than a girls talk when getting stuck in a research problem), Ryan Lekivetz, Chunfang Lin, Matt Pratola.

Finally, I also would like to thank all my friends here in Canada and back in Europe for their constant support and confidence in me. Very special thanks to my parents, my sister Julia and my grandmother for their love and support.

Contents

List of Figures

Chapter 1

Introduction

In the 1990's, with the increase of publicly available datasets, the field of Knowledge Discovery in Databases (KDD) emerged. KDD is the process of (semi-)automatic extraction of knowledge from databases which is valid, previously unknown and potentially useful [23]. This process consists of five major steps. The first three steps comprise the selection, preprocessing and transformation of the data. The transformed data forms the input to the selected Data Mining algorithm which then outputs the patterns. The interpretation and evaluation of these patterns provide us with knowledge. In this thesis we focus on the Data Mining process. Data Mining can be structured based on three dimensions: type of task, type of input data and type of application.

In Fayyad [23] the four main Data Mining tasks are described as: clustering; association rule and pattern mining; classification; and regression. Clustering refers to the task of grouping similar objects together. In association rule mining, people are concerned with finding interesting relations between features in large databases. The goal of classification is to learn a mapping from objects to specified classes, whereas in regression the mapping goes to real numbers. In this thesis, we developed a new clustering approach, a new pattern mining approach and a new statistical model.

The type of input data for a Data Mining task ranges from spatial data to web data, from transaction data to multi-relational data and network data. In this thesis, we focus only on multi-relational data and in particular on the subset of feature vector networks which we introduce later. Typically, multi-relational data are stored in a relational database consisting of several tables: entity tables and relationship tables. Entity tables store real world entities, such as people and books, whereas relationship tables contain information about the relationships between these entities, such as who has read which book.

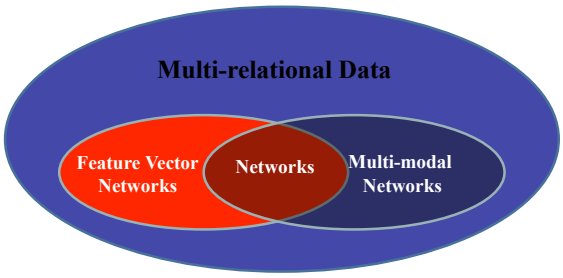


Figure 1.1: Multi-relational data and its subsets.

In Figure 1.1, a Venn diagram for multi-relational data and its subtypes is shown. In the context of this thesis, there are three relevant subtypes of multi-relational data: feature vector networks, networks and multi-modal networks. If the data consists of multiple entity types and the relationships between them, this data type is called *multi-modal networks* (right subset). *Networks* are modeled by a single entity table (without any features) and a single relationship table. Examples for network data include friendship networks and email exchange networks. If the entities have features, such age, gender or socio-economic status, we call these networks *feature vector networks* (FVNs) (red set). In most datasets, the network structure and feature vectors are dependent on each other, and mining them simultaneously promises to deliver more meaningful results than mining only one data type. An example of FVNs can be found in Figure 1.2. The entities (students) have a two-dimensional feature vector comprising of their hobby and research interest. For example, student A likes to play "soccer" and does research in the area of "logic". In terms of applications, there is a wide variety of applications for Data Mining in FVNs. Some of them will be introduced in the following.

1.1 Motivation

In many Data Mining applications, data is provided in the form of FVNs; social networks, such as Facebook and MySpace; biological networks, such as Protein-Protein interaction networks; road networks; the Internet, etc. In many social networks, the data contains not only the graph structure *e.g.*, implied by friendships, but other background information of the users, *e.g.*, their purchasing behavior or socio-economical features. One very popular Data Mining task is the identification of communities (clusters) in networks. While most work in this area has previously focused on the analysis of graph structures, feature vectors offer valuable information about the entities which can

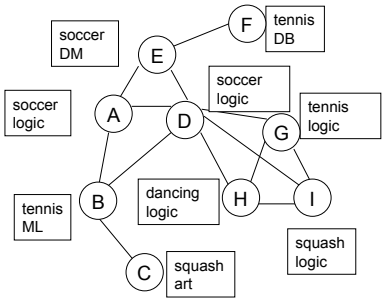


Figure 1.2: Example of social network of computer science students.

be useful in the mining process. One of the applications in this area is user segmentation with the goal of online advertising. The network structure, by itself, might not be sufficient for identifying the users' interests in a certain product or product group. On the other hand, considering solely the feature vectors (*e.g.*, past purchasing behavior), we would not be able to incorporate the behavior of a person's friends (network structure). Since people tend to be impacted by the purchases of their friends, this information is important.

Another application of clustering algorithms in FVNs is hotspot detection. In computational criminology, the goal of hotspot analysis is to identify high-crime areas (*hotspots*) surrounded by areas of relatively low crime rates (*coldspots*). Feature vectors could be the frequencies of certain crime types in a particular area over a given period of time. The road and public transportation networks constrain the space in which offenders move and also play a major role in the appearance of crime attractors and generators like pubs or needle sharing places [60]. Thus, two high-crime areas, which are geographically close, but not directly connected on the network, most likely have different attractors or generators and should not be merged into a single hotspot.

In computational biology, in Protein-Protein networks, these clustering algorithms can be used to identify so called *functional modules*. A functional module is a group of proteins that contributes jointly to the same cellular function. Modules can serve as the basis of computer-aided drug design.

In this thesis, we examine two related Data Mining tasks for FVNs that can be applied to several of the aforementioned problems. In particular, both approaches can be used for community identification, but each focuses on different aspects. In Chapter 4, we introduce a clustering algorithm for FVNs. This clustering algorithm is applied to both a collaboration network and other real world

datasets. The results are better than those obtained using other state-of-the-art methods. The introduced clustering algorithm finds a partitioning of the FVN, *i.e.*, each person/object is assigned to exactly one cluster. This is desired in a variety of applications including online advertising.

However, in other applications, such as module discovery or finding of small communities (*e.g.*, sport clubs or collaboration groups), this might not be desired. Communities can overlap since a person can be a member of a sports club *and* a research lab. Furthermore, the number of communities is often not known in advance. Typically, the members of a community know each other quite well, *i.e.*, there are many connections between them such that information can be exchanged and flow within the community. Also, members of a community are expected to have similar feature values in the subspace on which they are based, *e.g.*, features related to the personal or professional life. Finally, not every person has to be part of a community. Based on these requirements, in Chapter 5, we introduce a framework for identifying cohesive patterns. A cohesive pattern, is a subgraph which is dense, connected and in which the feature vectors are similar in a sufficiently large subspace. We show the effectiveness of the introduced algorithm by applying it to several real world datasets and evaluating its performance based on synthetic datasets. To the best of our knowledge, the dense graph mining algorithm of our approach is the first one which guarantees a complete result for density thresholds larger than $\frac{1}{3}$.

One of the problems we faced when working on the two afore-mentioned approaches, is the lack of publicly available datasets. To the best of our knowledge, a graph generator for FVNs does not exist. Since the network structure and feature vectors are dependent on each other, a separate generation of each data type is not a very satisfactory solution. This motivated us to work on a simulation model for FVNs. In Chapter 6, we introduce such a methodology. It is based on the previously introduced Latent Socio-Spatial Process (LSSP) model [47] and our extension, the n^*LSSP model.

We would like to emphasize the connection between the three research problems introduced in this thesis. All three problems deal with the analysis and simulation of FVNs. Furthermore, the first two research problems are alternative approaches for identifying communities. Depending on the application one is superior to the other. When working on these approaches, the lack of publicly available datasets proved challenging. Since none of the existing network generators were able to generate FVNs in which network structure and feature vectors are dependent on each other, there was a clear need for such a generator. Our simulation model is able to generate FVNs of various sizes in which network and feature vectors are dependent on each other.

1.2 Overview

In this thesis, we introduce three research problems in the area of Data Mining for FVNs: Connected X Clusters, Cohesive Pattern Mining and Simulation of Feature Vector Networks. The overview of this thesis, including the significance of each chapter, is listed below:

Chapter 2: Preliminaries We introduce basic definitions and notations used in this thesis.

Chapter 3: Related Work We review characteristics of (feature vector) networks and the related work in the area of clustering and graph mining.

Chapter 4: Connected X Clusters One of the major tasks in social network analysis is community identification. However, in most FVNs, the number of clusters (communities) is unknown. We propose an algorithm which determines this number and finds the corresponding clusters. The algorithm outperforms state-of-the-art methods. Parts of this work have been published in Moser et al. [56].

Chapter 5: Cohesive Pattern Mining Existing graph pattern mining algorithms produce a large number of patterns and cannot consider node-specific feature vectors. We propose a graph pattern mining approach which imposes constraints on the features. As a result the number of produced patterns is reduced and additional meaning is added to the identified patterns.
Existing algorithms for finding dense graphs had to rely on approximation algorithms. Our dense graph mining algorithm is the first one which guarantees completeness for density thresholds above $\frac{1}{3}$. Parts of this work have been published in Moser et al. [55] and Colak et al. [12].

Chapter 6: Simulation of Feature Vector Networks Existing generative graph models can only simulate networks without feature vectors. We propose a new methodology for simulating FVNs. Furthermore, we introduce an extension of the LSSP model which takes into account effects related to the network size.

Chapter 7: Conclusion We conclude this thesis.

Chapter 2

Preliminaries

This thesis is concerned with uni-modal networks. Uni-modal networks model a single type of entity. Social networks containing college students and their friendships are an example of uni-modal networks. Another example is Protein-Protein Interaction networks which are formal representations of the interactions between the different proteins contained in a cell. In particular, we are interested in uni-modal networks with node-specific feature vectors. We define this type of data as *feature vector networks*. Before this definition, we define the term graph and feature vector.

Definition 2.1 (Graph, Induced (Subgraph)) *A (undirected)* ***graph*** G *is a pair* $G = (V, E)$ *in which* $V = \{v_1, \dots v_n\}$ *denotes the node set and* $E \subseteq \{\{v_i, v_j\} \,|v_i, v_j \in V, v_i \neq v_j\}$ *the edge set. Let* $G = (V, E)$ *be a graph. The graph* $G[V'] = (V', E')$ *is called* ***a "by*** V'***" induced subgraph*** *(of* G*) if* $V' \subseteq V$ *and* $E' = \{\{v_1, v_2\} | v_1, v_2 \in V', \{v_1, v_2\} \in E\}$.

Based on this definition, the following short notations are used:

- $G + v$ refers to the graph induced by $V \cup \{v\}$, where v is a single node.
- $G + U$ refers to the graph induced by $V \cup U$, where U is a set of nodes.
- $G - v$ refers to the graph induced by $V \backslash \{v\}$.

Next, the term *connected graph* is defined.

Definition 2.2 (Connected Graph) *A graph* $G = (V, E)$ *is called* ***connected*** *iff for every pair of nodes* $v_i \in V$ *and* $v_j \in V$*, either* $\{v_i, v_j\} \in E$ *or there exists a set of nodes,* $v_1, \dots v_n$*,* $n \geq 1$*,* $v_k \in V$*,* $1 \leq k \leq n$*, such that* $\{v_i, v_1\}, \{v_1, v_2\}, \dots \{v_{n-1}, v_n\}, \{v_n, v_j\} \in E$.

This thesis, if not mentioned otherwise, only deals with undirected graphs. The term *graph* is mostly used in the theoretical and mathematical literature. Application-oriented papers tend to make use of the term *network*. We use these two terms interchangeably. Furthermore, the term *vertex* is used as a synonym for the term *node*. In some cases, a node is also called *data object*. We follow the social science literature and sometimes write *link* or *tie* instead of *edge*.

Next, the very basic definition of a feature vector is introduced.

Definition 2.3 (Feature Vector) *A **feature vector** is a d-dimensional, $d \geq 1$, vector $f(o) \in \mathcal{D}_1 \times \ldots \mathcal{D}_d$ that represents the properties of an object o.*

In the following, we combine the two definitions of a graph and a feature vector, such that the nodes in a graph have node-specific feature vectors. We call this type of graph a *feature vector network* and formally define it as follows.

Definition 2.4 (Feature Vector Network (FVN) (1)) *A **feature vector network (FVN)** is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F})$, in which $\mathcal{V}$ denotes the node set and $\mathcal{E}$ the edge set. The function $\mathcal{F} : \mathcal{V} \rightarrow \mathcal{D}_1 \times \ldots \times \mathcal{D}_d$ is called **feature function**. The set $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_d\}$ is called the **feature space** of $\mathcal{G}$. A subset $\mathcal{D}' \subseteq \mathcal{D}$ is called **feature subspace**.*

Social networks model people, their features and their interactions. An example of a social network represented as a FVN can be found in Figure 1.2. The nodes in that network have a two-dimensional feature vector attached. This data can be formally represented as

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F}),$$

where

$$\mathcal{V} = \{A, \ldots I\},$$

$$\mathcal{E} = \{\{A, B\}\{A, D\}, \ldots\},$$

$$\mathcal{D} = \{\{soccer, tennis, squash, dancing\}, \{DM, logic, ML, DB, art\}\}$$

and function $\mathcal{F}$, e.g., $\mathcal{F}(A) = (soccer, logic)$.

In Statistics literature, a FVN $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F})$ with $n = |\mathcal{V}|$ nodes is often represented by an $n \times n$-dimensional **sociomatrix** (or **adjacency matrix**) $\boldsymbol{y}$ with

$$y_{i,j} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

The d-dimensional feature vectors of the nodes are stored in a $n \times d$-dimensional covariate matrix $\boldsymbol{x}$. This leads to the following alternative definition:

Definition 2.5 (Feature Vector Network (2)) *$\mathcal{G} = (\boldsymbol{y}, \boldsymbol{x}, \mathcal{D})$ is called **feature vector network** iff $\boldsymbol{y}$ is an $n \times n$-dimensional sociomatrix and $\boldsymbol{x}$ an $n \times d$- dimensional covariate matrix. The variable $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_d\}$ contains the domains of each of the d rows in $\boldsymbol{x}$.*

Transformation of Feature Vector Networks into Weighted Graphs

When analyzing FVNs, the first thing that often comes to mind is a transformation of the FVN into a weighted graph. One way of doing this is to calculate the distance between the feature vectors of the two nodes participating in an edge. After this pre-processing step, existing methods for weighted graphs can be applied. Indeed, many approaches make use of this strategy and in some cases they deliver good results. However, the transformation of the feature vectors into distance values results in loss of information. Consider for example this very simple graph (see also Figure 2.1):

$$G = (\{X, Y, Z\}, \{\{X, Y\}, \{Y, Z\}\}, \{1, 2\}, \{F(X) = 1, F(Y) = 2, F(Z) = 1\}).$$

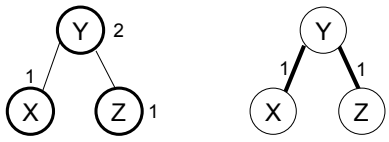


Figure 2.1: Transformation of FVN into weighted graph.

When considering a distance-weighted graph instead of a FVN, the information that the distance between node X and Y is the same as the distance between node Y and Z is retained. However, the information that X and Z have identical feature vectors is lost. Therefore, it makes sense to analyze a FVNs directly instead of its corresponding distance-weighted network.

Chapter 3

Related Work

In this chapter, an overview of characteristics in (feature vector) networks is provided in Section 3.1. Section 3.2 reviews the literature in the area of clustering, particularly network clustering and clustering of nodes in FVNs.

3.1 Characteristics of (Feature Vector) Networks

The analysis of FVNs is a fairly new research area. Most characteristics are explicitly specific to the network structure, *i.e.*, they do not take into account the feature vectors. However, since the characteristics are applicable to the network structure of a FVN, they are considered as well. The following section first introduces node-specific network measurements and then continues with an elaboration of global network measurements. At the end of this section, both the violation of the independence assumption of edges in networks and the characteristics of FVNs are discussed.

Node-Specific Network Measurements

Node-specific measurements can be used to compare the roles of nodes in a network. In the Internet domain, researchers are often interested in the importance of a single or small group of servers. In the case of their failure, an interruption of the Internet may ensue. In the following, let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph and $v \in \mathcal{V}$. The following definitions are taken from Wasserman et al. [72].

The **neighbors** $\mathcal{N}_i$ of a node v_i are defined as $\mathcal{N}_i = \{v_j : \{v_i, v_j\} \in \mathcal{E}\}$. The **degree** of node v, $deg(v)$, is defined as the number of v's neighbors, *i.e.*, $deg(v) := |\mathcal{N}_i|$. The **extended degree** of v corresponds to the number of v's neighbors' neighbors excluding v. The **minimum geodesic**

distance is the length of the shortest path between two vertices. The **clustering coefficient** of a node measures the local group cohesiveness. For a node v, it is defined as the ratio of the number of links between v's neighbors and the number of possible links between v's neighbors. The **betweenness** of a node v measures the number of shortest paths in the graph which go through v. This is a critical measurement in any transportation network, since people normally travel on the shortest path. In these networks, bridges tend to have a high betweenness. The **closeness** of a node v is the average distance from v to all other nodes. Intuitively, in a social network the closeness shows how well people are connected to the rest of the world.

Global Network Measurements

The clustering coefficient and betweenness can also be used as global network measurements by averaging over the corresponding values of all nodes in the network. The global clustering coefficient can be used as a measure of transitivity in the graph which is discussed below.

The **diameter** of a graph is the length of the longest shortest path in a network.

Two measurements often used to compare networks are motif count and degree sequence. The **motif count** is the histogram of the frequencies of different motifs in $\mathcal{G}$. A motif is a small graph with a well-defined structure. The **degree sequence/distribution** is also a histogram reflecting number/percentage of nodes with specific degrees.

When talking about clusters or communities in networks, the terms assortative and disassortative mixing are sometimes used, see Newman [59]. In **assortative mixing**, links between nodes within a given cluster are more likely to exist than links between nodes of different clusters. **Disassortative mixing** implies that links between clusters are more likely than links within a cluster. A dating network is an example for disassortative mixing. In contrast, friendship networks tend to be assortative. This thesis focuses only on assortative mixing.

Characteristics of Networks - Dependencies of Edges

In the following, characteristics of networks which violate the statistical independence assumption of the edges are discussed. There are three main characteristics: transitivity, clustering and reciprocity.

If v_i is connected to v_j, and v_j is connected to v_k, then it is more likely that v_i is also connected to v_k. This is reflected in a network by a larger than expected number of triangles and cliques. This phenomenon is called **transitivity**.

Clustering is closely related to transitivity. In the context of networks, the term **clustering** often

refers to the phenomenon that a subset of nodes exhibits a large number of within-group edges and relatively few edges outside of the group.

When analyzing a friendship network, it is expected that the friendship relationships are not independent. For example, if it is known that person i considers person j as a friend, then j is more likely to consider i as a friend than a randomly chosen person from the network. **Reciprocity** implies that $y_{i,j}$ and $y_{j,i}$ are statistically dependent. However, reciprocity impacts only directed networks which are not examined in this thesis.

As a result of the dependencies between the edges, most standard Data Mining or Statistical methods - which assume independent data as input - cannot be applied to FVNs. However, some existing approaches ignore some of these dependencies due to the complexity of the problem.

Characteristics of Feature Vector Networks

In the literature, there are two more characteristics specific to FVNs, namely heterogeneity in activity across nodes and homophily by attributes (feature vectors).

Heterogeneity in activity across nodes [45] refers to the observation that nodes with the same or very similar feature vectors are not required to have the same or a similar number of links. In a social network a *connector* [31], *i.e.*, a person who is very well-connected, can have the same interests or demographic features as a loner. As an example, consider the mayor of a city. Typically, a mayor knows many people and is very well connected. However, he can still have the same feature vector, *e.g.*, income and age, as a person who does not have many acquaintances. Some methods take this heterogeneity into account by using a random effect which models the *sociality* of a person.

The term **homophily by attributes** [53] refers to the observation that people tend to make friends with people who are similar to them in terms of their feature vectors. This phenomenon is also known as "birds of a feather flock together". Network models which consider feature vectors should take this into consideration.

3.2 Clustering

With the goal of identifying groups (clusters) whose members are similar to each other and dissimilar to the members of all other groups (clusters), clustering is one of the most common Data Mining tasks. The following section reviews the most popular clustering algorithms (organized by the type of their input dataset). First, the standard clustering algorithms are summarized. Some clustering algorithms do not consider the complete feature space, but only a subspace. These clustering

algorithms are called *subspace clustering methods* and are discussed next. Afterwards, network clustering is reviewed. Last, the literature for clustering of FVNs is summarized.

Clustering of Feature Vectors

The discovery of the true cluster structure in transactional data has been studied extensively. Three main approaches, partitioning, hierarchical and density-based methods, are widely used for this purpose. The most famous representative of partitioning clustering methods is k-means [50]. The objective of k-means is to partition the input set in k groups, such that the variance of the group elements is minimized. Hierarchical clustering methods, such as single-link, complete-link, average-link [40], CURE [32], BIRCH [77] and XMeans [63], achieve this goal by building a hierarchy of clusters to allow the exploration of clusters at any level of granularity. Density-based methods, such as DBSCAN [18], are able to find an arbitrary number of clusters based on certain constraints.

Subspace Clustering

Most clustering techniques consider all dimensions of feature vectors (the full space). However, in some high-dimensional data, many dimensions might be irrelevant, noisy or correlated. The aim of subspace clustering methods, *e.g.*, Parsons [61], is to identify the relevant dimensions in the clusters living within these dimensions. As a result, these clusters can overlap and have different relevant dimensions.

Network Clustering

As mentioned before, there are two different types of network clusters. *Assortative mixing* is the standard community structure and has more links within a community than between communities. However, there are some networks, like dating networks, where there tend to be more links between the communities (*e.g.*, male and female) than within. This thesis focuses only on assortative methods. A method which can be used to identify not only the type of mixing but also the clusters themselves can be found in Newman [59].

Graph Cut Methods

Research on graph partitioning has recently attracted much attention [71, 65, 37, 8]. The goal of graph partitioning is to divide a graph into several subgraphs by minimizing some objective function, such as cut-size, *i.e.*, the sum of weights of edges crossing between partitions [28, 26]. Methods

used for discovering an arbitrary number of graph partitions follow either a top-down or a bottom-up approach. In a top-down approach, in each round, one cluster is chosen and partitioned into two sub-clusters until certain conditions are satisfied. Different criteria, such as Ratio Cut [73, 34], Min-Max Cut [14], Q-function [58, 74], have also been studied for obtaining a natural partitioning of a graph. On the other hand, bottom-up approaches, such as ROCK [33] and CHAMELEON [43], start by assigning each node into its own cluster and merging similar clusters based on different criteria.

Shi and Malik [67] propose a global criterion, the *normalized cut*, as well as a grouping algorithm which recursively segments a graph into subgraphs. The normalized cut is also known as *conductance* [11] and is defined as

$$Ncut = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)},$$

where $cut(A,B)$ is the cut size between cluster A and B and $assoc(A,V)$ is the total connection size of nodes in A to all nodes in the graph which are denoted by V.

Dense Graph and Graph Pattern Mining

It might seem surprising to find the topic of dense graph mining in a section about clustering. One reason is that in many cases the goal of network clustering is to identify communities. There are many definitions of a community in the literature. Intuitively, a community is a group of people having some level of cohesion. Labs can be interpreted as communities within a department of a university. Considering a much larger network, *e.g.*, the social network of university students, departments themselves could be considered as communities. On the city level, the university can be seen as a community. The bottom line is that the definition and size of a community is dependent on the scale and application. In this work, we find that dense subgraphs can be considered as small communities.

Let $|V|$ be the number of nodes and $|E|$ be the number of edges in a graph $G = (V, E)$. There exist two very well accepted definitions of graph density. The first one is defined as the number of edges divided by the number of nodes, *i.e.*, $\frac{|E|}{|V|}$. In this thesis, the alternative definition is used:

$$d(G) = \frac{|E|}{\frac{|V|(|V|-1)}{2}}.$$

In this definition, the density is calculated as the fraction of the number of edges and the number of possible edges. In a graph whose density exceeds a certain threshold, the average degree of a node increases with the number of nodes. In comparison, in the first definition, the average degree remains

constant. Even when solving only approximately, the ability to find dense subgraphs in a large graph is a difficult combinatorial problem, *e.g.*, Feige et al. [24]. In the absence of a minimum support constraint, such algorithms typically forego the goal of finding all dense subgraphs and resort to heuristics that efficiently find some subset of these subgraphs. For example, Gibson et al. [30] present an algorithm based on a recursive application of shingling followed by a final clustering step.

A more relaxed version requires that the graph patterns are α quasi-cliques, *i.e.*, that every node has at least a specified percentage α of all possible edges within the pattern. Pei et al. [62] and Zeng et al. [76] propose new search space pruning strategies for efficiently mining all frequent and all closed frequent, respectively, α quasi-cliques.

Feature Vector Network Clustering

In the data mining community, clustering of FVNs has not received a lot of attention. This is most likely due to the lack of publicly available datasets. One of the first approaches is joint cluster analysis [17] which aims at partitioning a FVN into connected subgraphs. Similar to k-means, the optimization criterion of this method is to minimize the variance of feature vectors within a cluster.

The Co-Clustering method [36] defines an integrated distance function incorporating both the similarity of attribute data and the network's shortest path distance and then applies any distance-based clustering algorithm. Another integrated method that has been developed in the bioinformatics community is MATISSE [70], a probabilistic method that determines connected subnetworks in graphs (such as interaction networks) that exhibit high attribute (*e.g.*, expression) similarity, without enforcing clusters to be dense.

Chapter 4

Connected X Clusters

In this chapter, we introduce and formalize the **Connected X Clusters (*CXC*) problem**, the problem of discovering an a-priori unspecified number of clusters in FVNs. Clusters are assumed to be compact and distinct from their neighboring clusters. Furthermore, they form a connected subgraph. To efficiently solve the *CXC* problem, we introduce *JointClust*, an algorithm which adopts a two-phase approach. In the first phase, we find so-called *cluster atoms*. We provide a probability analysis for this phase which gives a probabilistic guarantee that each cluster is represented by at least one of the initial cluster atoms. In the second phase, these cluster atoms are merged in a bottom-up manner resulting in a dendrogram. The final clustering is determined by a novel objective function, called Joint Silhouette Coefficient. We evaluate *JointClust* on several real world datasets and show that its accuracy exceeds state-of-the-art comparison partners. Parts of this work have been published by Moser et al. [56].

4.1 Introduction

Ester et al. [17] proposed a partitioning clustering algorithm for FVNs, called Connected k-Center (CkC). Its goal is to discover k clusters. While partitioning clustering algorithms typically require the number of clusters as user input, it is well-known that prior knowledge on the number of clusters is often unavailable in real life applications. Previous research [63, 18, 40, 44] addresses the aforementioned problem for classical clustering models considering only the feature vectors. In this chapter, we study the problem of discovering an arbitrary (a-priori unspecified) number of so-called *(joint) clusters* in FVNs. Joint clusters are assumed to be compact and distinct from their neighboring clusters in terms of their feature vectors and form a connected subgraph. While in classical

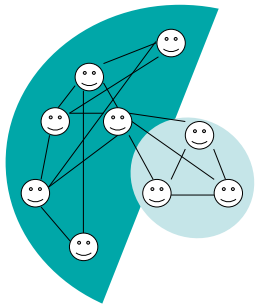


Figure 4.1: Example of social network partition.

clustering methods the neighborhood of clusters is (and can only be) defined in terms of feature vectors, in the context of FVNs it makes more sense to define the neighborhood in terms of graph data. To motivate this assumption, we introduce the following two running examples: community identification and hotspot analysis.

4.1.1 Motivation

Community Identification

A major task in social network analysis is to identify the underlying community structure of social networks. In this application, a person can be described by both feature vectors, such as demographic information or product preferences, and network data, such as social relations. Intuitively, a community is a connected subgraph of people with similar feature vectors, which are different from the ones of neighboring communities. The term *neighboring communities* refers to communities which have members who are directly connected to each other in the social network. An example (without feature vectors) can be found in Figure 4.1. Our goal is to identify communities whose members have similar feature vectors but significantly different values from the ones of the neighboring communities.

Hotspot analysis

In the context of criminology, the goal of hotspot analysis is to identify high-crime areas (*hotspots*) surrounded by areas of relatively low crime rates (*coldspots*). In this context, feature vectors represent the frequencies of certain crime types in a particular area over a given period of time. Edges

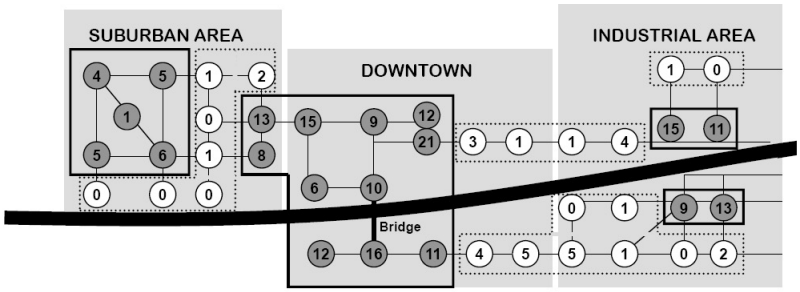


Figure 4.2: Example of road network.
Hotspots are in gray, coldspots in white.

represent connectivity information such as a road network or public transportation network. The network constrains the space in which offenders move and also plays a major role in the appearance of crime attractors and generators like pubs or needle sharing places [60]. Thus, two high-crime areas, which are geographically close, but not directly connected on the network, most likely have different attractors or generators and should not be merged into a single hotspot.

A simplified example with one-dimensional feature vectors can be found in Figure 4.2. Nodes are placed equidistantly along the road network, and the attached feature vectors measure the number of crimes in the surrounding area. In general, the crime rate in the downtown area is much higher than in other areas. This means that a feature-based clustering algorithm cannot find the crime hotspot in the suburban area, which has some areas with crime rates that are relatively high compared to their surrounding neighborhoods, but not especially high from a global point of view. The two parts of the downtown crime hotspot on the different sides of the river are connected via a bridge. When considering not only the feature vectors but also the network data, the hotspot can be correctly identified as one cluster.

4.1.2 Contributions

The discussion of the above applications motivates the need for finding an unknown number of connected subgraphs with feature vectors significantly different from the feature vectors of neighboring clusters (connected subgraphs). When discovering an a-priori unspecified number of clusters, the main challenge is to determine a suitable trade-off between model accuracy and model complexity in an unsupervised fashion using, *e.g.*, using an objective function. This objective function should also promote clusters which are compact and as distinctive as possible from their neighboring clusters.

Since none of the existing objective functions can deal with FVNs, we introduce a novel objective function called Joint Silhouette Coefficient. We also introduce an efficient clustering algorithm for FVNs, called *JointClust*, which detects the number of clusters. In the first phase, it identifies so-called cluster atoms, basic building components that cannot be split in the second phase. The number of initial cluster atoms is chosen so that there is a probabilistic guarantee that every joint cluster is represented by at least one of these initial cluster atoms. These cluster atoms are further iteratively refined as input for the second phase. In the second phase, in a bottom-up approach, cluster atoms are iteratively merged. The clustering with the highest objective function value is returned. The algorithm is evaluated on several real world datasets.

The main contributions of this chapter are:

- We introduce and formalize the problem of discovering an a-priori unspecified number of clusters in FVNs, called *CXC problem*.
- To solve the CXC problem, we propose a bottom-up algorithm, called *JointClust*, producing cluster atoms in the first phase, which are merged in the second phase.
- We provide probabilistic analysis demonstrating that with high confidence every joint cluster is represented by at least one of the cluster atoms generated in the first phase of *JointClust*.
- We experimentally evaluate *JointClust* on several real datasets.

Outline

First, we define the Connected X Clusters (*CXC*) problem in Section 4.2. We provide *JointClust*, a two-phase algorithm which solves the *CXC* problem, in Section 4.3. A probability analysis, which is used in the first phase of *JointClust*, is given in Section 4.4. *JointClust* is evaluated on several real world datasets in Section 4.5. The contributions of this chapter are summarized in Section 4.6.

4.2 Problem Definition

In this section, we introduce the Connected X Clusters problem, a clustering problem for FVNs which does not require the user to specify the number of clusters. The input of the problem is a FVN. The output is a cluster graph (defined below) which partitions the node set of the FVN into connected subgraphs (clusters). We start with some preliminary definitions: $\mathcal{G}$-connected and cluster

graph. Afterwards, we discuss model selection criteria and end this section with the introduction of the problem definition.

Preliminary Definitions

Definition 4.1 ($\mathcal{G}$-connected) *Let G_1 and G_2 be two non-overlapping, connected subgraphs of a FVN $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F})$. We call G_1 and G_2 **$\mathcal{G}-$connected**, denoted as $G_1 \leftrightarrow_{\mathcal{G}} G_2$ iff there exists at least one edge $\{v_1, v_2\} \in \mathcal{E}$, such that $v_1 \in G_1$ and $v_2 \in G_2$.*

Intuitively, the cluster graph of a FVN is a graph representing the result of a partitioning clustering algorithm applied to the FVN. It is formally defined as follows:

Definition 4.2 (Cluster Graph) *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F})$ be a FVN. Let $\{V_1, \ldots, V_k\}$, $k \geq 2$ be a partition of $\mathcal{V}$, i.e., $\mathcal{V} = V_1 \cup \ldots \cup V_k$ and $V_i \cap V_j = \emptyset$ for all $1 \leq i < j \leq k$.*
*A graph $CG_{\mathcal{G}}$ is called **cluster graph**, if $CG_{\mathcal{G}} = (\overline{V}, \overline{E})$, where $\overline{V} = \{V_1, ...V_k\}$ and $\overline{E} = \{\{V_i, V_j\} | V_i \leftrightarrow_{\mathcal{G}} V_j\}$ and if V_i, $i = 1, \ldots k$, is a connected component (**Connectivity Constraint**). The variable V_i is also called **cluster (node)**.*

One important property of a cluster is its center. Depending on whether or not the cluster center is required to be an observed data object, the following definitions are useful:

Definition 4.3 (Centroid) *Let V_i be a cluster. The **centroid** of V_i is defined as the component-wise average of the objects in V_i.*

Therefore, the centroid is not necessarily an observed data object.

Definition 4.4 (Medoid) *Let $V_i = \{v_{i,1} \ldots v_{i,n}\}$ be a cluster. The **medoid** of V_i is defined as the $v_{i,j}$, $1 \leq j \leq n$, which minimizes the distance to all other data objects in V_i.*

The medoid is an actual data object.

Criteria for Model Selection

The number of possible cluster graphs for a given FVN is very large. In order to compare different cluster graphs objectively, a model selection criterion is needed. Many existing objective functions explicitly penalize models with larger numbers of clusters by calculating a tradeoff between clustering accuracy and model size (number of clusters), *e.g.*, Pelleg et al. [63] which uses the Bayesian

Information Criterion. Other objective functions implicitly penalize large models, *e.g.*, the Silhouette Coefficient [44] that compares for every data object the distance to the centroid of the assigned cluster with the distance to the centroid of the second-closest cluster. The Silhouette Coefficient judges the clustering quality independent of the number of clusters. For every data object i, it compares the distance of i to the centroid of the cluster to which it is assigned with the distance to the closest centroid (from the set of all centroids excluding the one from the cluster to which it is assigned). Formally, it is defined as

$$s = \frac{1}{n} \sum_{i=1}^{n} \frac{b(i) - a(i)}{max\{b(i), a(i)\}},$$

where $a(i)$ is the distance from data object i to the centroid c of the cluster to which it is assigned, and $b(i)$ is the distance to closest centroid (not considering c). Therefore, the Silhouette Coefficient is always between -1 and 1. The higher the value, the better the clustering and vice versa.

In the following, we introduce an adapted version which can be used for FVNs, called *Joint Silhouette Coefficient*. One property of the *CXC* problem is that the feature vectors of a cluster are not supposed to be as distinctive as possible from *all* other clusters, but only from all *connected* clusters. For this reason, the Joint Silhouette compares the distance of each data object to the assigned cluster center with the average distance to the centers of all neighboring clusters.

Definition 4.5 (Joint Silhouette Coefficient) *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F})$ be a FVN and $CG_{\mathcal{G}} = (\overline{V}, \overline{E})$ be a cluster graph. Furthermore, let a be the cluster to which node i is assigned and $a(i)$ the distance of i to the centroid of a. Let $b_E(i)$ be the average distance to the centroids of all $\mathcal{G}$-connected clusters of a. The **Joint Silhouette Coefficient** S is defined as*

$$S = \frac{1}{|V|} \sum_{i \in V} \frac{b_E(i) - a(i)}{max\{b_E(i), a(i)\}.}$$

For example, in Figure 4.2, the feature vectors of the two hotspots in the industrial area are quite similar. However, we do not want to penalize the model for having two clusters (hotspots) with similar values, if the clusters are not connected. Since the Joint Silhouette Coefficient considers only the distance to the neighboring clusters, the existence of these two clusters does not negatively influence the value of the Joint Silhouette Coefficient. Therefore, the Joint Silhouette Coefficient allows us to take into account only the *local* neighborhood of a cluster.

Connected X Clusters Problem

Clustering is the process of partitioning data objects into groups according to some similarity measure, such that data objects within one group are similar to each other and dissimilar to data objects from other groups [54]. This definition is not directly applicable to FVNs, in which not only feature vectors, but also network data carry important information. We require a cluster to be a connected subgraph. The data objects of a cluster are still required to be as similar as possible to each other, but not as dissimilar as possible to *all* clusters, but only to the *connected* clusters. Ester et al. [17] introduced the CkC problem, the problem of identifying such clusters. In their problem definition the number of clusters had to be specified by the user. However, in many real world applications this number is not known in advance. The *CXC* problem does not require this number to be specified and is formally defined as follows:

Definition 4.6 (Connected X Clusters (CXC) Problem) *Given a FVN $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{D})$ and an integer value m, the minimum cluster size. The **Connected X Clusters Problem (CXC)** is to find a cluster graph $CG_G = (\overline{V}, \overline{E})$ such that*

1. *each cluster (node) $v \in \overline{V}$ contains at least m data objects (**minimum size constraint**), and*
2. *CG_G maximizes the Joint Silhouette Coefficient.*

Intuitively, *CXC* aims at identifying a partition of the FVN, such that each partition forms a connected subgraph containing at least m objects. The partition should guarantee that the clusters are compact and their cluster centers are as distinctive as possible from the ones of all connected clusters. Our assumption is that the input graph is connected. If it is not connected, the problem definition is applied separately to every maximum connected component.

In many applications, such as social network analysis, a community is expected to have a minimum size, m. For example, a sports club having 10 people might be not very meaningful. The minimum size constraint ensures that the size of each cluster exceeds this minimum size. However, if desired, m can be set to 1.

4.3 Algorithm

In Section 4.3.1, we introduce *JointClust*, a two-phase algorithm solving the *CXC* Problem. In Section 4.3.2 we analyze its runtime.

4.3.1 *JointClust*

In the following, we propose *JointClust*, an algorithm solving the *CXC* Problem as defined in Definition 4.6. *JointClust* is based on the idea of adapting X-means and the Silhouette Coefficient to FVNs.

JointClust is a bottom-up heuristic algorithm, which consists of two phases. In the first phase, we determine cluster atoms, which are merged bottom-up in the second phase to determine the final clustering. Each final cluster consists of at least one cluster atom. In the first phase, after the initialization part, there is a probabilistic guarantee that each cluster is represented by at least one cluster atom (Theorem 4.2). The refinement part improves the quality of the identified cluster atoms. The second phase merges bottom-up the so far identified cluster atoms with the goal to find the clustering which maximizes the Joint Silhouette Coefficient (Definition 4.5). We return the clustering with the highest Joint Silhouette Coefficient from the generated dendrogram. We discuss *JointClust* now in detail; its pseudo code can be found in Algorithm 1.

Algorithm 1 *JointClust*

1: INPUT: FVN, m, $nrIt$
2: // FIRST PHASE - Initialization Part
3: $reqNum$ = reqNrCentr(size(FVN)/m)
4: $iniCentroids$ = choose $reqNum$ random nodes from FVN
5: $iniClusterAtoms =$ genClusterAtoms($iniCentroids$)
6: $iniClusterAtoms$ = mergeSmallClusters($iniClusterAtoms$)
7: // FIRST PHASE - Refinement Part
8: $clusterAtoms_0 = iniClusterAtoms$
9: **for** (i = 1 to $nrIt$) **do**
10: $centroids_i =$ getMedoids($clusterAtoms_{i-1}$)
11: $clusterAtoms_i =$ genClusterAtoms($centroids_i$)
12: $clusterAtoms_i$ = mergeSmallClusters($clusterAtoms_i$)
13: **end for**
14: // SECOND PHASE
15: **return** $finalClusterGraph =$ mergeAtoms(clusterGraph($clusterAtoms_{nrIt}$))

The input of *JointClust* is a FVN, FVN, and the minimum cluster size m, which is necessary to find meaningful clusters. However, this is not a restriction of the generality of *JointClust*, since this value can be set to 1. The last parameter is $nrIt$ and determines how often the refinement part is repeated. This number of iterations is a common parameter in most optimization algorithms; it can be set to a constant value, *e.g.*, to the value 10.

First Phase

The goal of the first phase is to identify cluster atoms. A *cluster atom* is a basic building component consisting of a connected component which cannot be split in the second phase. The first phase consists of an initialization part and a refinement part. To ensure each cluster is represented among the cluster atoms, the number of initial centroids is set higher than the maximum number of possible clusters (which is bound by the total number of data objects divided by the minimum size of a cluster). First randomly $reqNum = reqNrCentroids(\lceil size(FVN)/minClusterSize \rceil)$ initial centroids ($iniCentroids$) are sampled from the input graph, see Theorem 4.2 in Section 4.4 for a detailed explanation. As confidence a standard value of 0.95 is used. Next, the method `genClusterAtoms()` is applied to this set.

At the beginning, each of the initial centroids, $iniCentroids$, is assigned to its own cluster. We perform a breadth-first search starting from the nodes that are already assigned to clusters. A node can only be assigned to a cluster, if it is directly connected to at least one of the nodes contained in this cluster. Nodes are greedily assigned in the described fashion by always choosing the node that has the smallest distance (in terms of the feature vectors) to the existing cluster centroids.

After having assigned each node to a cluster, the resulting clusters are guaranteed to be connected components. The cluster graph is constructed as follows: for each cluster, a cluster node is created. Two cluster nodes are connected, if any two of their contained nodes are connected, see also Definition 4.2. The generated cluster nodes form the cluster atoms.

Since the input graph was assumed to be connected, we know that the resulting cluster graph is connected as well. To ensure that the minimum size constraint is fulfilled, too small cluster atoms are joined with their closest connected cluster atoms. The function `mergeSmallClusters()` takes care of this.

The initialization part of the first phase guarantees that there is at least one centroid (and its corresponding cluster atom) placed within each cluster with a certain probability, see Section 4.4. To improve the quality of the final cluster atoms used as input for the second phase, the cluster atoms are iteratively refined.

Similar to k-means, the refinement part of the first phase starts with calculating the medoids of the previously discovered cluster atoms. These medoids are used as initial centroids in the `genClusterAtoms()` function. These refined centroids, $centroids_i$, are of better quality than the random ones used in the initialization part of the first phase, since they are centrally located (in terms of feature vectors and the location within the subgraph) in their cluster atoms. While the initial

centroids come with a probabilistic guarantee that at least one of them is chosen from each cluster, these centroids could be located on the border (in terms of their feature vectors and/or the location within the subgraph) of a cluster. The refined centroids avoid any of these problems that the initial - randomly chosen - centroids may have. To further improve the quality of the centroids and their corresponding cluster atoms, the methods `getMedoids()` and `genClusterAtoms()` are run for $nrIt$ times.

Second Phase

The result of the first phase is a cluster graph. In the current case a set of cluster atoms, which partitions the node set of the input graph. Each cluster atom is as compact as possible and each true cluster is represented by at least one cluster atom. The goal of the second phase is to find the final clustering by bottom-up merging one pair of similar neighboring cluster atoms in every step, using the Joint Silhouette Coefficient as objective function to choose the best clustering in a particular step.

The subroutine `mergeAtoms()` (Algorithm 2) has as input a cluster graph. Each edge in the cluster graph represents a possible merge, since it preserves the connectivity and minimum cluster size constraints. At the beginning, the Joint Silhouette Coefficient of the input graph is calculated (Line 2). We calculate the Joint Silhouette Coefficient for all possible merges (edge e) of a given cluster graph $\texttt{clusterGraph}_k$ and pick the one with the highest value (Line 7). Afterwards, the chosen cluster graph is updated reflecting the merge; the number of clusters is reduced by one (Lines 9 and 10). We continue with this strategy until only two cluster nodes remain. Like the conventional Silhouette Coefficient, the Joint Silhouette Coefficient is a non-monotone function. Therefore, the merging has to continue until only two clusters are left (the Silhouette Coefficient is only defined for at least two clusters). The cluster graph with the highest Joint Silhouette Coefficient is returned.

4.3.2 Runtime Complexity

In this section, we analyze the runtime of *JointClust* and compare it with the runtime of a bottom-up algorithm starting with singleton clusters: clusters containing only a single node.

The runtime of the **First Phase** depends on the runtime of `genClusterAtoms()`. Its runtime is $O(reqNum\ nrIt\ n^2)$, where n is the number of data objects and $reqNum$ the number of initial centroids. Merging too small clusters can be computed in $O(reqNum^2)$, since there are at most $reqNum$ clusters and each cluster has at most $reqNum - 1$ connected neighbors. The parameter

Algorithm 2 Sub-routine mergeAtoms()

```
1: INPUT: clusterGraph_k, k (= clusterGraph_k.NumberOfNodes)
2: S_k = getJointSilCoeff(clusterGraph_k)
3: while (k ≥ 2) do
4:    for (each edge e in clusterGraph_k) do
5:       S_{k-1,e} = getJointSilCoeff(clusterGraph_k.merge(e))
6:    end for
7:    e_max = edge leading to highest S_{k-1,e}
8:    S_{k-1} = S_{k-1,e_max}
9:    clusterGraph_{k-1} = clusterGraph_k.merge(e_max)
10:   k = k − 1
11: end while
12: return clusterGraph_i with the highest S_i
```

$nrIt$ is a constant. Since $n \gg reqNum$, the value $reqNum$ cab be considered as a constant as well. Therefore the runtime of the first phase is $O(n^2)$.

In the **Second Phase**, the number of cluster nodes of the input graph is bound by $l = \lceil n/m \rceil$. An undirected graph with l nodes has at most $O(l^2)$ edges. In each iteration of the while-loop all possible merges of two clusters are tried, *i.e.*, $O(l^2)$. This implies a decrease of the number of clusters by 1 in each iteration. Therefore, there are $O(l)$ iterations of the while-loop. The total runtime of the second phase results hereby in $O(l^3)$.

The **overall runtime** of *JointClust* is $O(n^2)$, since $l < reqNum \ll n$ and $reqNum$ can be considered as a constant. A purely bottom-up algorithm takes $O(n^3)$. The use of the first phase of *JointClust* reduces this cubic runtime to a quadratic one.

4.4 Analysis

The objective of *JointClust* is to learn the number of clusters and to identify these clusters. Similar to k-means, *JointClust* also starts with a set of initial centroids. Next, we analyze the influence of the initialization on k-means and on *JointClust*. Similarly to k-means, the result of *JointClust* is also dependent on choice of the initial centroids. In *JointClust* we determine the appropriate number of initial centroids required for ensuring that at least one initial centroid is placed within each *joint cluster* with a user-defined probability. The primary idea is to select more than the required number of initial centroids such that there is a probabilistic guarantee of having at least one initial centroid per joint cluster. These centroids, together with their neighborhood serve as *cluster atoms*.

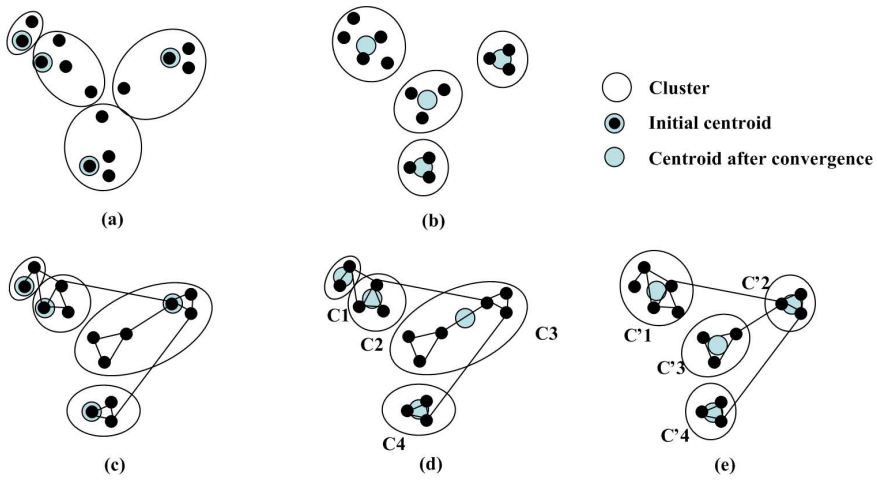


Figure 4.3: Various initializations.
(a) Undesired initialization for 4-means. (b) Desired result of 4-means. (c) Example of initialization of *JointClust*. (d) Undesired result of *JointClust*. (e) Desired result of *JointClust*.

4.4.1 Influence of the Initialization on k-means

The k-means algorithm [15, 5] is known to converge to a local minimum of its distortion measurement (total squared distance). In each iteration, the average squared distance decreases or remains the same. It is also well-known that k-means is sensitive to its initialization. Strategies for finding sophisticated initializations were introduced *e.g.*, by Bradley et al. [7]. They suggest to repeatedly subsample and smooth to get a better initialization. However, in most cases k-means is initialized several times with different centroids where the initialization resulting in the smallest distortion measurement is chosen. Figure 4.3(a) shows an example of an undesired initialization. The desired solution of 4-means can be found in Figure 4.3(b).

4.4.2 Influence of the Initialization on *JointClust*

Similar to k-means, *JointClust* is also dependent on the initialization. In the first phase, *JointClust* greedily assigns data objects to the initial centroids under consideration of the connectivity constraint. Any network clustering algorithm considering the connectivity constraint has a very restricted search space. Therefore, the initialization is critical. As an example, see Figures 4.3(c) to 4.3(e). Figure 4.3(c) shows the FVN with randomly chosen initial centroids and 4.3(d) the result after convergence. Based on the initialization, the desired clustering of Figure 4.3(e) cannot be achieved.

To overcome the limitation of the constrained search space, we draw a higher number of initial centroids than the maximal possible number of clusters. Next, we show, how to determine that number of initial centroids.

4.4.3 Determination of the Number of Initial Centroids

In the following, we prove two theorems. The goal of the first theorem is to calculate the likelihood for given k clusters and k trials (= flips of coins), that exactly one data object is drawn from each cluster. The second theorem states how many trials are necessary in order to draw at least one data object per cluster with a given confidence value (probability).

Theorem 4.1 *Given a dataset V, let $V_1, ...V_k$ be the clustering (partition) of V and $|V_i|$ the number of data objects contained in cluster V_i. Let m be the minimum size of a cluster.*
The probability that we draw exactly one element of each cluster in k trials is between $\left(\frac{m}{|V|}\right)^k k!$ and $\frac{k!}{k^k}$.

Proof Let E_i^k denote the event that in k trials we draw at least one data object from cluster V_i. Let k be the number of clusters. The probability of drawing exactly one data object from each cluster in k trials is:

$$p = P(E_1^k \wedge ... \wedge E_k^k) = P(\wedge_{i=1}^k (E_i^1))k! = \prod_{i=1}^k P(E_i^1)k! \quad (1)$$

If we want to draw at least one data object per cluster (having in total k clusters) and we have only k trials, then this is equivalent to drawing exactly one data object per cluster. Since the order is irrelevant, the resulting term is multiplied with $k!$. The probability for choosing a data object out of an arbitrary cluster is the cluster size divided by the total number of data objects. Therefore $P(E_i^1) = \frac{|V_i|}{|V|}$. Let $m \leq \frac{|V|}{k}$ be an arbitrary minimum cluster size. In the following, we calculate the upper and lower bound of p, using (1).

$$p = k! \prod_{i=1}^k \frac{|V_i|}{|V|} \leq \frac{k!}{|V|^k} \left(\frac{\sum_{i=1}^k |V_i|}{k} \right)^k = \frac{k!}{k^k} \quad (2)$$

$$p = k! \prod_{i=1}^k \frac{|V_i|}{|V|} \geq k! \prod_{i=1}^k \frac{m}{|V|} = \left(\frac{m}{|V|} \right)^k k! \quad (3)$$

$$\left(\frac{m}{|V|} \right)^k k! \leq p \leq \frac{k!}{k^k} \quad (4)$$

In equation (2) the inequality of the arithmetic and geometric mean [10] is used. This inequality states the following:

$$\sqrt{\prod_{1}^{k} x_i} \leq \frac{\sum_1^k x_i}{k},$$

for non-negative real number $x_1, \ldots x_k$. Equation (3) makes use of $|V_i| \geq m$. Inequality (4) follows immediately from (2) and (3). Therefore, the probability for drawing one data object per cluster, given k clusters, in k trials is between $\left(\frac{m}{|V|}\right)^k k!$ and $\frac{k!}{k^k}$.□

To illustrate these probabilities, let us look at the following example. Let $|V| = 1000$ be the size of the graph, $m = 100$ be the minimum size and $k = 10$ be the number of clusters, then $p = 0.00036$. Thus, the chances of placing one of the initial centroids in each cluster is very small. In order to address this problem, we propose the following strategy: to find at most k clusters, we use more than k initial centroids. The exact number is determined using the following theorem.

Theorem 4.2 *Given a dataset V with $n = |V|$ data objects, let m be the minimum size of a cluster. Let $V_1, ...V_l$ be the clustering (partition) of V and $|V_i|$ be the number of data objects in cluster V_i. Let $k = \lceil \frac{n}{m} \rceil$. In order to choose each cluster at least once with a probability of p, it suffices to draw $reqNum = \lceil k \ln \frac{k}{-\ln p} \rceil$ data objects.*

Proof As in the previous proof, let E_i^s denote the event that cluster V_i is chosen at least once within s trials. Therefore, $\neg E_i^s$ indicates that V_i has not been chosen in s trials.

$$P(\neg E_i^s) = \left(1 - \frac{|V_i|}{n}\right)^s \leq \left(1 - \frac{1}{k}\right)^s \leq e^{-\frac{s}{k}}$$
$$\iff P(E_i^s) \geq 1 - e^{-\frac{s}{k}}$$

Let $P(E_{all}^s)$ be the probability that all k clusters were chosen at least once within s trials. We have

$$P(E_{all}^s) = P(\wedge_{i=1}^k E_i^s) = \prod_{i=1}^{k} P(E_i^s) \geq (1 - e^{-\frac{s}{k}})^k \geq 4^{-ke^{\frac{-s}{k}}},$$

since $\forall x \leq \frac{1}{2} : (1-x)^{\frac{1}{x}} \geq \frac{1}{4}$. In order to select with probability p each cluster at least once in s trials,

$$4^{-ke^{\frac{-s}{k}}} \geq p$$

are required. In order to determine the smallest s satisfying the inequality, this results in $reqNum = s = \lceil k \ln \frac{k}{-\ln p} \rceil$, since $reqNum$ must be an integer.□

Confidence	0.900	0.950	0.990	0.999
Nr. of Initial Centroids	61	68	85	108

Table 4.1: Number of initial centroids required for $n = 1000$ and $m = 100$

This theorem is similar to the famous coupon collector problem, *e.g.*, Feller et al. [25]. The values in Table 4.1 refer to the same example as described before ($n = |V| = 1000$ and $m = 100$). *JointClust* uses Theorem 4.2 in order to determine the required number of initial centroids so that there is a probabilistic guarantee that each cluster is represented by at least one cluster atom.

4.5 Experiments

JointClust is evaluated on two different types of real world datasets: social network data and image data. Both types of data can be represented as FVNs. Our first dataset is a co-author network extracted from DBLP and citeseer. We compare the accuracy of *JointClust* with that of X-Means [63] (feature-based clustering) and normalized cut [68] (network-based clustering), see Section 4.5.1. In Section 4.5.2, we evaluate JointClust on image data.

4.5.1 Social Network Data

The social network dataset consisted of a co-authorship network. Next, its generation is described.

Dataset Generation

The co-authorship network was generated based on the two well-known scientific literature digital databases: citeseer[1] and DBLP[2]. Newman [57] showed that this type of databases reflect the properties of general social networks very well. The chosen papers were written between 2000 and 2004. They belonged to three different research areas: Theory, Machine Learning and Databases & Data Mining. These papers were written by 1,942 distinctive authors. An author's class label was determined by the majority class of the papers they have written. Roughly the same number of authors belonged to each research area.

The "term frequency inverse-document frequency" (tf-idf) [64] is a state-of-the-art method for text representation. This method increases the importance of a word proportionally to the number

[1] http://citeseer.ist.psu.edu/

[2] http://www.informatik.uni-trier.de/~ley/db/

of times this word appears in the document (abstract of paper) but downgrades words which occur frequently. We employed tf-idf to the 603 most important keywords occurring in the abstracts of the corresponding papers and used them as author-specific feature vectors. The co-authorship of at least one paper was used to generate an edge between the corresponding authors.

Result

JointClust was used to cluster the authors in the network based on the keywords of their papers and their co-authorship relationships. As mentioned before, authors were labeled with the majority area of their papers. Theses labels are considered as *true labels*. In order to evaluate the clustering, first the majority of the true labels in each cluster was assigned to each node in the respective cluster. Then the accuracy was calculated as the fraction of true positives and the total number of nodes.

As comparison partners, we used the implementation[3] of Normalized Cut [68] a graph partitioning algorithm, and X-Means [63], a feature-based clustering algorithm, which does not require the number of clusters as input. We used the Java implementation of X-Means provided by Weka 3.5.5[4]. All clustering algorithms used the Cosine Distance, the standard distance measurement for text documents. *JointClust* requires the minimum size of a cluster as input parameter, which was set to 100. Scientific communities with fewer members do not seem significant enough. We use the default number of iterations, *i.e.*, 10. X-Means was run with the default parameters. However, X-Means is not able to consider the co-authorship relations, since it can only deal with feature vectors and the Normalized Cut is not able to consider the feature vectors.

The results can be found in Table 4.2. *JointClust* achieved an average accuracy of 76.7%, whereas X-Means resulted in 61.4% (both methods averaged over 20 repetitions) and the normalized cut achieved 68.1%. *JointClust* identified the correct number of clusters in all repetitions, whereas X-Means always discovered 4 clusters (instead of 3) and was, in each repetition, substantially below the accuracy of *JointClust*. Our implementation of the normalized cut required the number of clusters as input parameter; it was set to the expected number, namely 3. These results confirm that feature vectors and network data indeed contain complementary information which helps to identify the clusters.

[3] http://www.seas.upenn.edu/~timothee/software/ncut_multiscale/ncut_multiscale.html

[4] http://sourceforge.net/projects/weka/

	Accuracy	Identified Number of Clusters
X-Means	61.4%	4
Normalized Cut	68.1%	*
JointClust	76.7%	3

Table 4.2: Social Network Dataset - expected number of clusters is 3; our implementation of normalized cut required the number of clusters as input, we set it to the expected number of clusters (3).

4.5.2 Image Data

The goal of *image segmentation* is to partition a digital image into multiple regions (sets of pixels) [66]. In the following, we use *JointClust* as a basic image segmentation algorithm, to demonstrate its application to image data, and not to compare ourselves with state-of-the-art approaches in the image processing community. Our main reason to use image data was its similarity to crime data, to which we did not have access to.

Similarity of Image Data to Hotspot Data

Privacy concerns make the access to crime databases almost impossible. To compensate for the unavailability of this type of data, we run *JointClust* on image data. On a conceptual level, these two data types are very similar. Most modern North American cities have a grid-like road network. The resulting graph structure of an image, in which neighboring pixels are connected, is therefore very similar to the network of these cities. The crime rate corresponds to the extracted feature values of each pixel. The hotspots and coldspots correspond to image segments. As in hotspot analysis, in image segmentation it makes sense to assume a segment (cluster) to be compact in terms of its feature vectors and distinct from its neighboring segments (clusters).

In this experiment, the feature vectors are given by the three-dimensional Lab color space value (L stands for lightness and a and b for color-opponent dimensions) [39] and three texture features. To extract these values from the input images, we used the MatLab program *blobworld*[5] provided by the university of Berkeley. The details of their feature extraction algorithm and their image retrieval algorithm, blobworld, are described by Carson et al. [9]. Unlike their approach, we did not consider the coordinates of the pixels as additional feature vectors, but used neighborhood relationships of the pixels to generate the network data. The edges were generated by connecting all pairs of neighboring

[5] http://elib.cs.berkeley.edu/src/blobworld

(directly adjacent) pixels. The goal of blobworld is image retrieval, which is completely different from ours, and a comparison against their method was not meaningful. Unfortunately, the images had no ground truth, so that we could not quantitatively measure the clustering accuracy in this application. Instead, we qualitatively measured the clustering accuracy of selected images by visual inspection.

Technical details

The Euclidean distance was used as distance measurement between the feature vectors. The minimum size of a cluster was set to 1,000 pixels, the total number of pixels per image was 19,712. The number of iterations was set to 10. We evaluated *JointClust* on several images of the famous Corel stock photo collection.

Results

Figure 4.4 shows the results of *JointClust* on three different images. In part (a) of these figures, a gray-scaled version of the original coloured image is shown. In part (b), we present the cluster atoms identified in the first phase. The number of cluster atoms still exceeds the number of true clusters. However, we can see that each true cluster is already represented by at least one cluster atom. Typically, the second phase succeeds in correctly merging the atoms to detect the true clusters. The last part (c) depicts the final result of the second phase which merges the cluster atoms in a bottom-up manner and chooses the final result based on the value of the Joint Silhouette Coefficient. Note that *JointClust* considers only connected regions as clusters. (Note, due to gray-scale postprocessing of the image, we colored different clusters with the same shade in order to make the different clusters as recognizable as possible.)

In Figure 4.4, the rose is very well discovered by *JointClust*. It also determines the number of clusters correctly. As we can see in part (b) of Figure 4.4, the first phase produces many cluster atoms, which are correctly merged in the second phase. The rose and tiger image in part (c) of Figure 4.4 show two clusters. The tiger is completely discovered. All cluster atoms are correctly merged in the second phase. The Joint Silhouette Coefficient, which only compares neighboring clusters, is able to merge the cluster atoms in the desired way.

The third image depicts a bridge, which is a very challenging image. The shade is responsible for breaking the bridge into two parts. The connectivity constraint is responsible for connecting the cables with the bridge and splitting the background into two parts. However, even if human beings

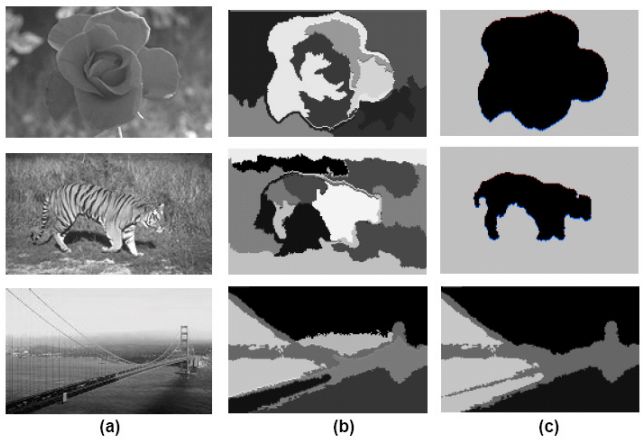


Figure 4.4: Image data results.
(a) Original Image. (b) Result after First Phase. (c) Final Result.

were asked to partition this image, the answer would not be unanimous: some might merge the background with the cables, others might not. Since the land on the left hand has colors very similar to the bridge, it was falsely merged by *JointClust*. In order to recognize such challenging objects, more sophisticated image processing algorithms are necessary, which consider background knowledge such as the shape of certain object types. As mentioned above, the goal of these experiments was not to compete with these methods, but to show further applications of *JointClust* to real world data.

4.6 Conclusion and Future Work

In this chapter, we introduced a new problem definition, called *Connected X Centers (CXC)*. Its goal is to find a partition of the input graph into clusters. Clusters are connected subgraphs whose feature vectors are similar to each other and dissimilar to the ones of neighboring clusters. To efficiently solve the *CXC* problem, we presented *JointClust*, an algorithm which adopts a two-phase approach. *JointClust* determines cluster atoms in the first phase, which are merged in a bottom-up strategy in the second phase. Our experimental evaluation on several real life datasets, a social network and several images, shows that *JointClust* is able to identify the correct number of clusters and the clusters themselves with high accuracy. The comparison with X-Means, a clustering algorithm, which cannot consider network data, shows that network data indeed carry important and complementary information while improving the accuracy significantly.

This research suggests several directions for future research. We proposed a distance-based measure, the Joint Silhouette Coefficient, to quantify the trade-off between model complexity and accuracy. Probabilistic measures, with their known advantages and disadvantages, deserve further attention.

The connectivity constraint requires a simple connectedness of the subgraph, that means within a cluster there exists at least one path between any pair of nodes. This type of connectivity might not be sufficient in some applications. For example, in biological applications, it is sometimes desirable to impose a density constraint on the clusters. However, in applications that deal with high-dimensional feature vectors, it might be worthwhile to investigate a subspace clustering approach for FVNs to avoid the *curse of dimensionality*. In the next chapter, we integrate the density requirement and the subspace idea into a new problem definition. Furthermore, *JointClust* results in a partition of the input graph. While this is desirable in certain applications, other applications may want overlapping clusters. We consider this in the problem definition introduced in the next chapter as well.

Chapter 5

Cohesive Pattern Mining

In this chapter, we introduce the novel problem of mining cohesive patterns from FVNs, which combines the concepts of dense subgraphs and subspace clusters. A cohesive pattern is a dense and connected subgraph that has homogeneous values in a sufficiently large feature subspace. We argue that this problem definition is natural in identifying small communities in social networks and functional modules in Protein-Protein interaction networks. We present the algorithm *CoPaM* (Cohesive Pattern Miner), which exploits various pruning strategies to efficiently mine all maximal cohesive patterns for density threshold larger than $\frac{1}{3}$. We are the first ones who provide an algorithm for finding the complete set of all dense subgraphs for the aforementioned density threshold. Our theoretical analysis proves the correctness of *CoPaM*. Our experimental evaluation on real life social network and biological network data demonstrates that *CoPaM* finds meaningful patterns and is more accurate than other state-of-the-art methods. Our experiments on synthetic datasets confirm the scalability of our proposed algorithm.

5.1 Introduction

Graphs provide a natural representation of important real life networks, such as social networks and biological networks. While earlier analysis methods often focused on graph properties such as degree distribution, diameter and simple graph patterns, more recent analysis methods aim at identifying more sophisticated patterns and structures in graphs. Online social network data can be analyzed, for example, to detect communities that can be used for more targeted delivery of online advertisements. In systems biology, researchers want to find functional modules in protein interaction networks which can serve as the basis of computer-aided drug design.

Most of the existing methods, such as graph partitioning [68] and quasi-clique finding [62], work on graph data only. However, in many applications the data come in the form of FVNs. In these FVNs, often feature vectors and edges contain complementary information, *i.e.*, the links cannot be derived from the feature vectors nor vice versa. In such scenarios the simultaneous use of both data types promises more meaningful and accurate results. Joint cluster analysis [17] aims at partitioning a graph with feature vectors into connected components whose nodes have similar feature vectors. Shiga et al. [69] introduce a spectral clustering method which partitions graphs with feature vectors. While these approaches can exploit FVNs, they cannot ensure that the discovered clusters are dense and connected, since they partition the entire graph. As a consequence, the identified clusters cannot overlap, which is a desired property in many contexts.

Integrating the concepts of dense subgraphs and subspace clusters, we introduce the novel problem of **mining cohesive patterns**. We define a cohesive pattern as a connected subgraph whose density exceeds a given threshold. Furthermore a cohesive pattern has homogeneous feature vectors in a sufficiently large subspace. Different from graph partitioning methods and similar to frequent-pattern mining methods, cohesive patterns can overlap and do not have to cover the entire dataset. Moreover, the number of patterns does not need to be specified in advance. A major criticism of pattern mining is the large number of patterns produced. In our case, the number of dense and connected subgraphs can be extremely high. Integrating constraints on the feature vectors reduces the number of patterns considerably and adds additional meaning to the identified patterns. Our algorithm *CoPaM* effectively prunes the search space by simultaneously using the constraints on the feature vectors, density and connectivity, and has therefore to consider only a small portion of the large number of possible subgraphs.

5.1.1 Motivation

The following two applications from social network analysis and systems biology are used to motivate our problem definition.

Social Network Analysis

In social network analysis, one of the most important tasks is the identification of communities [71], *i.e.*, groups of people that have strong social interactions and share some interest. Communities like sports clubs or research groups have the following characteristic properties: Their members know each other quite well, *i.e.*, have many edges between them such that information can be exchanged

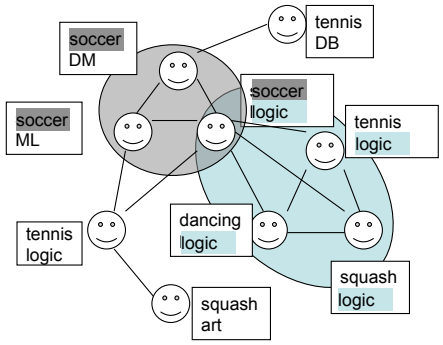


Figure 5.1: Example of a social network of CS students containing two communities.

and flow within the community. Members of a community are expected to have similar feature vectors in the subspace on which they are based, *e.g.*, features related to the personal or professional life. Communities can overlap, since a person can be a member of a sports club and a research lab. The number of communities is not known in advance. Finally, not every person has to be part of a community. Figure 5.1 shows an example of a social network of computer science students, wherein students are associated with two features, their favourite sports and their study focus. The first community consists of soccer players, united by their favorite sports, and the second one contains students who share logic as their study focus. One student belongs to both communities. Other students are not part of any community, because they do not share any interest (research or sports) with their friends (connected nodes).

Systems Biology

In systems biology, one application of our problem definition is the identification of functional modules. Functional modules are groups of genes that are involved in a specific cellular process. In the literature, initial attempts were restricted to the use of a single data type such as gene expression or protein interaction data. However, each of these data types describes only one specific aspect of the cellular system and fails to characterize the system as a whole. Most cellular functions are carried out by a group of proteins, that highly interact with each other, but loosely interact with the rest of the proteins. Therefore, a functional module forms a dense and connected component in the interaction network. Functional modules are also characterized by similar expression patterns (feature vectors) of their genes (that code the proteins of the module). Similar expression data are

not required in all conditions (dimensions) of the feature vectors, but only in a subset, because many proteins perform different functions in different tissues during different developmental stages. Two approaches, which successfully combined both data types, can be found in Ulitsky et al. [70] and Hanisch et al. [36]. We compare our method with these approaches.

5.1.2 Contributions

The main contributions of this chapter are as follows:

- We introduce the novel problem of **mining cohesive patterns**, which integrates the concepts of mining dense subgraphs and subspace clustering.
- We develop the algorithm *CoPaM* that efficiently mines the set of all maximal cohesive patterns for the density threshold $\alpha \geq \frac{1}{3}$.
- We provide a theoretical analysis giving insights into the difficulty of the problem and prove the correctness of *CoPaM*.
- We run experiments on social network, biological and synthetic datasets demonstrating the meaningfulness of cohesive patterns and show the efficiency and scalability of *CoPaM*.

Overview

The rest of this chapter is organized as follows. In Section 5.2, the problem definition of mining cohesive patterns is introduced. The algorithm *CoPaM* is presented in Section 5.3. A theoretical analysis of *CoPaM* can be found in Section 5.4. Section 5.5 reports the results of our experimental evaluation. Extensions of *CoPaM* are elaborated in Section 5.6. This chapter is concluded with a summary and interesting directions for future research in Section 5.7.

5.2 Problem Definition

In this section, we define the problem of mining cohesive patterns in FVNs. In the following, $\mathcal{G}$ denotes the complete input FVN and G denotes a subgraph of $\mathcal{G}$. For an overview of notations used in this chapter, see Table 5.1. We are interested in mining *cohesive patterns*, *i.e.*, subgraphs G of $\mathcal{G}$ with certain properties. First, cohesive patterns are required to be connected. Second, their density $d(G)$ needs to exceed a given threshold. The density is defined as the cliquishness, which is defines

Table 5.1: Notations

$\mathcal{G}$	Input FVN
$G = (V, E, D)$	By V induced subgraph of $\mathcal{G}$
$d(G)$	$= \frac{2\|E\|}{\|V\|(\|V\|-1)}$ (density of G)
$deg(v)$	Degree of node v, $v \in V$
s	Subspace cohesion function
θ_s	Subspace cohesion threshold
θ_d	Dimensionality threshold
α	Density threshold
G^{merge}	Merging candidate

as the number of edges divided by the number of possible edges. Third, the features of the nodes of G are required to be cohesive in some feature subspace. In order to formalize this last constraint, the definition of a subspace cohesion function is necessary.

Definition 5.1 (Subspace cohesion function) *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F})$ be a FVN. A **subspace cohesion function** s is a Boolean function*

$$s : \mathcal{P}(\mathcal{V}) \times \mathcal{P}(\mathcal{D}) \times \Re \rightarrow \{true, false\},$$

where $\mathcal{P}$ denotes the power set. Therefore, the function s has as input a subset V of the node set $\mathcal{V}$, a subset D of the feature space $\mathcal{D}$, and a real number, θ_s, such that

$$[[s(V, D, \theta_s) = true] \wedge [\nexists D' : \mathcal{D} \supset D' \supset D : s(V, D', \theta_s) = true]] \Rightarrow$$

$$[s(V, D'', \theta_s) = true \Rightarrow D'' \subseteq D],$$

*where D is called **maximal feature subspace**. Based on this definition D is uniquely determined for any subgraph. Furthermore, s is assumed to be anti-monotone, i.e.,*

$$s(V, D, \theta_s) = true \Rightarrow s(V', D', \theta_s) = true \forall V' \subset V, D' \subset D.$$

*The threshold θ_s is called **subspace cohesion threshold**.*

In summary, a subspace cohesion function is an anti-monotone function which implies a uniquely determined maximal feature subspace. Note that *CoPaM* is also able to handle anti-monotone functions which do not have a uniquely determined maximal feature subspace, such as order-preserving submatrices [2] as shown at the end of Section 5.3. To illustrate the subspace cohesion function, consider again the example in Figure 5.1. All nodes in a community have the same value in at least one dimension. Formally,

$$s(V, D, \theta_s) = true$$

if

$$\forall v \in V, d \in D : F_d(v) = c$$

for some value c. A further example is the one used in Section 5.5.2. There, the function s is elected as:

$$s(V, D, 1, 25) = [\forall d \in D : |\max\{F_d(v), v \in V\} - \min\{F_d(v), v \in V\}| \leq \theta_s = 1.25]$$

The function $F_d(v)$ denotes the value of v's feature vector in dimension d.

So far, there has not been a restriction on the size of D, *i.e.*, in the worst case D is empty and s true for any node set V. To prevent such a case and to enforce some stricter cohesion, we constrain the size of D in the definition of a cohesive pattern. This definition is next and is the most critical definition of this chapter.

Definition 5.2 (Cohesive pattern) *Given a FVN $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F})$ and the following parameters:*

- *subspace cohesion function s,*
- *subspace cohesion threshold θ_s,*
- *dimensionality threshold θ_{dim} and*
- *density threshold α.*

*An induced subgraph $G = (V, E, D)$, $V \subset \mathcal{V}, E = \{v_1, v_2 | v_1, v_2 \in V, \{v_1, v_2\} \in \mathcal{E}\}, D \subset \mathcal{D}$, is called **cohesive pattern** if it satisfies the following three constraints:*

- ***Subspace cohesion constraint:*** *G is homogeneous in $D \subseteq \mathcal{D}$, i.e., $s(V, D, \theta_s) =$ true and $|D| \geq \theta_{dim} \geq 1$.*
- ***Density constraint:*** *$d(G) := \frac{2|E|}{|V|(|V|-1)} \geq \alpha$. (In this case G is also called α-dense.)*
- ***Connectivity constraint:*** *G is connected.*

*The density constraint, the subspace cohesion constraint, and the connectivity constraint together are called **cohesive pattern constraint (CP constraint)**. Furthermore, an edge is called **cohesive** if the induced subgraph of its corresponding nodes fulfills the CP constraint, otherwise **non-cohesive**.*

In the following, the terms *pattern* and *subgraph* are used interchangeably. We are particularly interested in mining maximal cohesive patterns which are defined as follows:

Definition 5.3 (Maximal cohesive pattern) *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F})$ be a FVN and $G = (V, E, D)$ be a cohesive pattern. The pattern G is called **maximal cohesive pattern**, if there does not exists $V' : V \subset V'$ such that the graph $G' = (V', E', D')$ induced by V' is also a cohesive pattern. Furthermore, we require D to be maximal, i.e., there does not exists $D' : D \subset D'$, such that $G = (V, E, D')$ is a cohesive pattern as well.*

This definition leads to the following problem definition in which we want to mine maximal cohesive patterns.

Cohesive pattern mining problem

Let $\mathcal{G}$ be a FVN, function s be a subspace cohesion function, threshold θ_s be a subspace cohesion threshold, threshold θ_{dim} be a dimensionality threshold and threshold α be a density threshold, the **Cohesive Pattern Mining Problem** is to mine the set of all maximal cohesive patterns of $\mathcal{G}$ wrt. the aforementioned parameters.

In the following, we define several properties of cohesive patterns and of nodes which are used by *CoPaM*.

Definition 5.4 ((Maximally) expanded-by-one cohesive pattern) *A cohesive pattern G, $G = (\{v_1, \cdots v_n\}, E, D)$ is called **expanded-by-one** if there exists at least one permutation $\tau = (v_{i_1}, \ldots v_{i_n})$ over the nodes of G that induces a sequence $(\{v_{i_1}, v_{i_2}\}, \ldots, G - \{v_{i_{n-1}}, v_{i_n}\}, G - v_{i_n}, G)$, such that all graphs in this sequence are cohesive patterns wrt. D. If a cohesive pattern G cannot be extended by any neighboring node without violating the CP constraint, G is called **maximally expanded-by-one cohesive pattern**.*

The reasoning behind this definition is the following: a graph G is expanded-by-one if it can be iteratively generated from a single node by adding one connected node at a time, such that all resulting patterns are cohesive patterns.

Not all *maximally expanded-by-one* patterns are *maximal cohesive patterns*, as the example in Figure 5.2 demonstrates. For $\alpha = 0.41$, the white nodes form a maximally expanded-by-one pattern, since adding one of the black nodes results in a density below α. However the pattern induced by the white nodes is not maximal: the graph containing all black and white nodes is maximal.

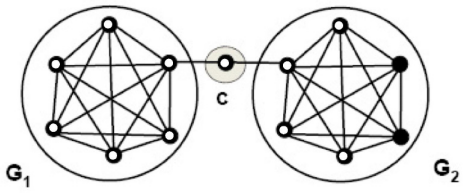


Figure 5.2: Example of cohesive pattern containing α-critical node for $\alpha = 0.41$.
Node c is α-critical. The patterns $G_1 + c$ and $G_2 + c$ are merging candidates. The subgraph induced by the white nodes is maximally expanded-by-one, since adding one of the black nodes results in a density below α.

Furthermore, the complete graph is also not expanded-by-one, since the only node which can be removed without violation of the density constraint is c. The node c is a bridge node which is defined in the following:

Definition 5.5 (Bridge node) *Let G be a graph, a node c in G is called **bridge node** if $G - c$ is a disconnected graph.*

Since c is a bridge node, in $G - c$ the connectivity constraint is violated. In Section 5.4, we show that this case can only occur for $\alpha < \frac{1}{2}$. Cohesive patterns which are not expanded-by-one, still have a very nice property. They can be decomposed into three subgraphs, namely two merging candidates and one expand-by-one part. The merging candidates are defined in the following.

Definition 5.6 *A cohesive pattern $G_1^{merge} = (V, E, D)$ is called **merging candidate** if it consists of a $\frac{1}{2}$-dense graph G_1, $|G_1| \geq 4$, and a simple path P, $|P| \geq 1$, such that $\exists p \in P, v \in G_1$, such that $(p, v) \in E$ and $\forall p' \in P, p' \neq p, (p', v') \notin E, \forall v' \in G_1$. Furthermore, if $|P| = 1$, $deg(p) < deg(v) + 1 \forall v \in G_1$, otherwise $deg(p) < deg(v) \forall v \in G_1$. The node with the smallest degree in G_1^{merge} is called **merging candidate node**.*

For example, in Figure 5.2, two of the merging candidates are $G_1 + c$ and $G_2 + c$. This concept of merging candidates is needed by *CoPaM*.

Definition 5.7 (α-removable node, α-critical) *Given cohesive pattern $G = (V, E, D)$, $c \in V$. The node c is called **α-removable node** if $G - c$ is α-dense. A cohesive pattern G is called **α-critical**, if every α-removable node is a bridge node. The α-removable nodes in an α-critical graph are called **α-critical nodes**.*

The graph in Figure 5.2 is α-critical, since the only α-removable node is the bridge node c. Therefore, the node c is called α-critical node.

Complexity of the Cohesive Pattern Mining Problem

We briefly analyze the complexity of finding all maximal cohesive patterns. It is known that finding the maximum clique (MAXCLIQUE) in a graph is NP-complete (see Karp [42]). In our case, a clique is a cohesive pattern if its feature vectors are homogeneous. Having an oracle which provides a solution for the cohesive pattern mining problem, the solution can be verified in polynomial time. Thus, our problem is in NP. Since MAXCLIQUE can be reduced to the cohesive pattern mining problem, the problem is NP-complete. However, we expect the size of the largest clique to be constant due to the sparseness of the graphs under consideration. Therefore, in practice this part of the problem is feasible.

In case of a trivial subspace cohesion function, which is true for any input, our problem can be reduced to the problem of counting all cliques in a graph. This is known to be #P-Complete [27]. Again, this theoretical worst case does not typically occur in real world datasets, and the runtimes reported in Section 5.5 show the practicality of *CoPaM*.

5.3 Algorithm

In this section, we introduce the algorithm ***CoPaM*** **(cohesive pattern miner)**, which solves the cohesive pattern mining problem. This two phase algorithm adopts a level-wise bottom-up pattern enumeration, *i.e.*, the search space of cohesive patterns of size n is based on the cohesive patterns of size $n-1$. In Section 5.3.1, two non-integrated approaches are introduced, whereas Section 5.3.2 discusses the algorithm.

5.3.1 Baseline Approaches

A non-integrated cohesive pattern mining approach first finds all connected and dense patterns and checks the subspace cohesion constraint afterwards. An alternative algorithm might first find all subsets of nodes which are cohesive in a sufficiently large subspace and check the density and connectivity constraints afterwards. In the experimental section, we compare *CoPaM* with these two non-integrated baseline pattern mining algorithm, which are explained in the following:

Baseline 1

Algorithm: *CoPaM*-based connected and dense pattern generation.
Postprocessing: Checking against subspace cohesion constraint.
Details: First, all connected and dense patterns are generated using *CoPaM*, see Section 5.3.2. A trivial subspace cohesion constraint is used which is true for any pattern. Then *CoPaM* is run with this constraint in order to generate all connected and dense patterns. Second, as postprocessing, all candidate patterns not satisfying the subspace cohesion constraint are filtered out.

Baseline 2

Algorithm: Apriori-based subspace clustering.
Postprocessing: Checking against connectivity and density constraints.
Details: First, all subsets of nodes (clusters) whose feature vectors satisfy the subspace cohesion constraint are generated. We use an apriori-based approach which exploits the anti-monotonicity of the subspace cohesion function to do this. Second, from the identified clusters, the ones which do not fulfill the connectivity and density constraints are filtered out.

Remark

For both baseline methods a maximality check is necessary.

Alternative Strategy

One commonly used strategy which could be considered as a third baseline is the following: A second graph, called similarity graph, is constructed by thresholding the feature vector similarity. Both graphs are mined simultaneously. However, similarity on the complete feature space leads to significant loss of information, as shown in Chapter 2. This is why, in such applications, subspace clustering methods have been proven to outperform full space clustering methods. However, the strategy of using a similarity graph cannot be adapted to subspace clustering, since constructing similarity graphs for all possible subspaces is intractable.

5.3.2 *CoPaM*

The pseudo code of *CoPaM* can be found in Algorithm 3; the first phase of the algorithm (**expand-by-one**) in Algorithm 4 and the second phase (**merge**) in Algorithm 5. The input of *CoPaM* is a

Algorithm 3 *CoPaM*: Cohesive Pattern Miner

INPUT: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{A}), \alpha, s, \theta_s, \theta_{dim}$
OUTPUT: maximal cohesive patterns
PREPROCESSING: remove non-cohesive edges from $\mathcal{G}$
for all (connected components $C_i = (V_i, E_i)$ in $\mathcal{G}$) **do**
  // FIRST PHASE: EXPAND-BY-ONE
  $currCohesivePatterns_i \leftarrow \emptyset$
  $mergingCand_i \leftarrow \emptyset$
  for all (edges $e = \{v_1, v_2\} \in E_i$) **do**
    $G_e \leftarrow (\{v_1, v_2\}, \{e\}, D)$
    $currCohesivePatterns_i$.add(G_e)
  end for
  $currCohesivePatterns_i, mergingCand_i \leftarrow$ Expand-by-one($currCohesivePatterns_i$)
  if ($\alpha < \frac{1}{2}$) **then**
    // SECOND PHASE: MERGE
    $mergedPatterns_i \leftarrow$ merge($mergingCand_i$)
    $currCohesivePatterns_i$.add(Expand-by-one($mergedPatterns_i$))
  end if
end for
$currCohesivePatterns = \cup_i currCohesivePatterns_i$
MAXIMALITY CHECK: remove non-maximal cohesive patterns from $currCohesivePatterns$
return $currCohesivePatterns$

FVN $\mathcal{G}$ and the following parameters: a density threshold α, where $\frac{1}{3} < \alpha \leq 1$, a subspace cohesion function s, a subspace cohesion threshold θ_s and a minimum number of dimensions θ_{dim}. The output is the set of all maximal cohesive patterns.

The algorithm starts with a **preprocessing phase**, in which non-cohesive edges are removed from the input graph, since they cannot be part of any cohesive pattern. In this reduced graph, all connected components are identified. In the following, each of these components is analyzed separately, applying the first and second phase of *CoPaM*.

The first phase (expand-by-one) starts off by initializing the variables $currCohesivePatterns_i$ (containing the current cohesive patterns) and $mergingCand_i$ (containing the current merging candidates). Next, for each edge a cohesive pattern is created. Each pattern contains also its maximal feature subspace D. Patterns are added to the set of current cohesive patterns. Afterwards, the Expand-by-one method is applied to this set.

The **Expand-by-one** method (Algorithm 4) takes as input cohesive patterns of size two and returns maximally expanded-by-one cohesive patterns. Let $level$ denote the number of nodes of the current cohesive patterns (initially it is two), see also Figure 5.3. In each level, a cohesive patterns of size $level$ is expanded by all neighboring nodes (one at a time) obtaining patterns of size $level + 1$.

Algorithm 4 *CoPaM* (first phase): Expand-by-one

```
1: INPUT: Queue currCohesivePatterns
2: OUTPUT: maximally expanded-by-one cohesive patterns, merging candidates
3: Queue resultSet ← ∅
4: Set mergingCand ← ∅
5: while (G ← currCohesivePatterns.pop()≠ NULL) do
6:    level ← size(G)
7:    if (α < 1/2) then
8:       if (isMergingCandidate(G)) then
9:          mergingCand.add(G)
10:      end if
11:      G.isMaximal← true
12:   end if
13:   for all (neighboring nodes v of G) do
14:      if (G + v fulfills CP constraint) then
15:         if (NOT currCohesivePatterns.contains(G + v)) then
16:            currCohesivePatterns.add(G + v)
17:            G.isMaximal← false
18:         end if
19:      end if
20:   end for
21:   if (G.isMaximal) then
22:      resultSet.add(G)
23:   end if
24: end while
25: return resultSet, mergingCand
```

For every expanded pattern, the respective maximal cohesive feature subspace is determined. If an expanded pattern $G + v$ fulfills the *CP* constraint (Algorithm 4, Line 14), then G is not maximal and can be replaced by $G + v$ in the candidate set $currCohesivePatterns$, otherwise G is added to the result set. If α is smaller than $\frac{1}{2}$, the algorithm also checks whether the current pattern is a merging candidate and potentially adds it to the set of merging candidates. After having considered all patterns of a certain level (size), the algorithm moves to the next level until all patterns are maximally expanded-by-one. The variable $currCohesivePatterns$ which holds the patterns, is implemented as a queue. Therefore, the expand-by-one method uses a breadth-first search. The advantage of this search strategy is that at any point in time only cohesive patterns of two levels have to be kept in memory - this reduces the amount of memory needed substantially.

The expand-by-one phase generates only expanded-by-one cohesive patterns, which follows directly from its definition. We show in the next section that if $\alpha \geq \frac{1}{2}$, all cohesive patterns are indeed expanded-by-one cohesive patterns. Therefore, the first phase finds all cohesive patterns for $\alpha \geq \frac{1}{2}$. If α is between $\frac{1}{3}$ and $\frac{1}{2}$ a second phase, which takes the merging candidates as input,

is required. In this second phase, called **merge phase**, the search space is restricted to merging candidates which were identified in the first phase. If $\alpha < \frac{1}{3}$ then the algorithm is not guaranteed to be complete. However, in applications such as social network analysis and systems biology typically α is larger than $\frac{1}{3}$, in most cases even larger than $\frac{1}{2}$.

Algorithm 5 *CoPaM* (second phase): Merge

```
1: INPUT: hashtable mergingCand
2: OUTPUT: α-critical graphs
3: result ← ∅
4: for all (G1, G2 ∈ mergingCand which have only the merging cand. node in common) do
5:     if (( G1 ∪ G2) is α-critical) then
6:         result.add(G1 ∪ G2)
7:     end if
8: end for
9: return result
```

The **merge phase** (Algorithm 5) takes a set of merging candidates and joins two merging candidates if they have the same merging candidate node. This is efficiently supported by using a hashtable as an index structure, whose keys are the node IDs of the merging candidate node. As shown in Section 5.4, only graphs which overlap by *exactly one* node (a merging candidate node) have to be considered. This phase returns only α-critical cohesive patterns. The expand-by-one phase has to be applied again to these patterns.

The goal of the final **maximality check** is to remove cohesive patterns which are not maximal. This postprocessing step is necessary, since the *CP* constraint is not anti-monotone, see Section 5.4.1. This is different from frequent itemset mining, where the anti-monotonicity property guarantees that only maximal patterns are found. In order to efficiently filter out the maximal patterns, we

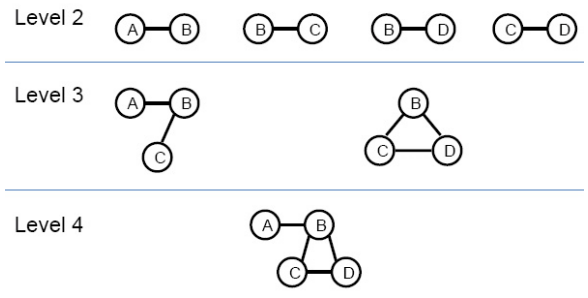


Figure 5.3: Illustration of expand-by-one phase.

use the following algorithm: The result set of *CoPaM* is implemented as a queue, such that any new element is added at the beginning, and during this maximality check the elements can be retrieved in reverse order of their insertion in constant time. Furthermore, the maximality check makes use of a hashtable and an array. The keys of the hashtable are the node IDs. The buckets store all maximal cohesive patterns which contain this particular node. A second data structure, an array, records the number of elements in each bucket. All cohesive patterns identified by *CoPaM* are inserted into the hashtable in opposite order of their creation, *i.e.*, starting with the largest pattern. A new pattern P is added as follows:

- Find smallest bucket $B_{smallest}$ to which P has to be added, using the array information.
- Test whether P is subset of any of the elements in $B_{smallest}$.
  If P is not subset, add P to the hashtable.
  If P is subset, then discard it, since P is not maximal.

Not uniquely determined maximal feature subspace

Some applications benefit from a subspace cohesion function that does not uniquely determine the maximal feature subspace. This is the case for order-preserving submatrices [2] often used in biological applications. This type of subspace cohesion function can easily be integrated into *CoPaM*. The algorithm has to be slightly adapted to store *all* (instead of one) maximal feature subspaces with the current pattern.
This concludes the description of *CoPaM*. In the following section we prove its correctness for $\alpha \geq \frac{1}{3}$.

5.4 Correctness of *CoPaM*

The correctness of the first phase seems intuitive. However, it turns out that it requires several proofs to show its correctness. We start by analyzing the monotonicity properties of the *CP* constraint and show that it is not anti-monotone. This makes it very different from other pattern mining algorithms which heavily make use of this property. In Section 5.4.2, we show that the *CP* constraint is loose anti-monotone (defined in Section 5.4.1) for $\frac{1}{2} \leq \alpha$ and we prove that the expand-by-one phase finds all cohesive patterns in this case. For $\frac{1}{3} \leq \alpha < \frac{1}{2}$, the *CP* constraint is not loose anti-monotone, but it has some nice properties that allow for finding all cohesive patterns by merging the merging candidates which were identified in the expand-by-one phase, see Section 5.4.3.

5.4.1 Monotonicity Properties

In the following, the terms anti-monotone and loose anti-monotone are defined:

Definition 5.8 (Anti-monotone) *Let a graph G satisfy constraint C, then C is called **anti-monotone** if **all** subgraphs of G satisfy C.*

Definition 5.9 (Loose anti-monotone) *Let G be a graph G of size n satisfying constraint C. We call C **loose anti-monotone** [6] if there **exists at least one** subgraph of G of size $n-1$ which fulfills C as well.*

Frequent pattern mining algorithms, such as the frequent item set mining algorithm a-priori [3], make use of the anti-monotonicity property of the support. Unfortunately, the ***CP* constraint is not anti-monotone**. In other words, for a given cohesive pattern there is no guarantee that each node can be removed such that the remaining pattern is still connected. A counter example is shown by the cohesive pattern in Figure 5.2 on page 42, where the removal of c (the only α-removable node) disconnects the graph.

In the following, we analyze the monotonicity properties of the connectivity and density constraint. Note that the subspace cohesion constraint is anti-monotone by definition.

Loose anti-monotonicity of the connectivity constraint

Lemma 5.1 *Let $G = (V, E)$ be a connected graph of size at least 2. There exist two distinct nodes $v_1, v_2 \in V$ such that $G - v_1$ and $G - v_2$ are connected.*

Proof We use induction on the number of nodes in G. If G does not contain any bridge node, then we are done. Otherwise, let v be a bridge node in G. Then $G - v$ consists of $l > 1$ connected components $G_1, \ldots G_l$. If G_1 contains only one node, then this node is not a bridge node in G. Suppose G_1 has more than one node. By induction hypothesis, there are two distinct nodes u and w in G_1 such that they are not bridge nodes in G_1. If vu or vw, say vu, is not an edge in E, then u is not a bridge node in G, therefore $G - u$ is connected. Otherwise uv and wv are edges in G, then $G - u$ is connected. Thus, there is a node in G_1 which is not a bridge node.
A similar argument is used for G_2.□

Proposition 5.1 *The connectivity constraint is loose anti-monotone.*

Proof Lemma 5.1 states that in any graph, there exists two nodes, such that their removal does not disconnect the graph. Therefore, there exists one such node. This implies that the connectivity constraint is loose anti-monotone. □

Loose anti-monotonicity of the density constraint

Proposition 5.2 *The density constraint is loose anti-monotone.*

Proof Let $G = (V, E)$ be an α-dense graph, $v \in V$, and $G' = G - v = (V', E')$. We distinguish:

- $\exists v \in V$: $deg_G(v) < \lceil \alpha(|V| - 1) \rceil$.
  $\Rightarrow |E'| = |E| - d_G(v) \geq \alpha \frac{(|V|-1)(|V|-2)}{2} = \alpha \frac{(|V'|)(|V'|-1)}{2}$
- $\forall v \in V : deg_G(v) \geq \lceil \alpha(|V| - 1) \rceil$
  Let $v \in V$ be the node with minimum degree m, $m \geq \lceil \alpha(|V| - 1) \rceil$ in G.
  $\Rightarrow |E'| \geq \frac{|V|m}{2} - m \geq \frac{1}{2}(|V| - 2)m \geq \frac{1}{2}(|V| - 2)\alpha(|V| - 1) = \alpha \frac{|V'|(|V'|-1)}{2}$

Therefore, G' is α-dense.□

Summary:

- The subspace cohesion constraint is anti-monotone. (By Definition 5.1.)
- The density constraint is loose anti-monotone. (By Proposition 5.2.)
- The connectivity constraint is loose anti-monotone. (By Proposition 5.1.)

5.4.2 Correctness of *CoPaM* for $\frac{1}{2} \leq \alpha$

After having analyzed the monotonicity properties of each of the constraints separately, we analyze now the simultaneous satisfaction of them for $\alpha \geq \frac{1}{2}$.

Theorem 5.1 *Simultaneous satisfaction of the connectivity and density constraints is loose anti-monotone for* $\alpha \geq \frac{1}{2}$.

Proof Let $G = (V, E, D)$ be connected and α-dense for $\alpha \geq \frac{1}{2}$. Assume the only α-removable nodes in G are bridge nodes. Let $b \in V$ be one of them. Then $G - b$ contains at least two connected components G_1 and G_2. There is at least one node in G_1, say v_1, whose removal does not disconnect G (application of Lemma 5.1 to $G_1 + v$). Since v_1 was not α-removable, $deg_G(v_1) > \alpha(|V| - 1)$.

Therefore, G_1 contains more than $\alpha(|V|-1)-1+1 \geq \frac{1}{2}(|V|-1)$ nodes. Using a similar argument for G_2 results in G having more than $|V|$ nodes, which is a contradiction. Therefore, there exists a node $v \in V$, such that $G - v$ is connected and α-dense.□

Algorithmic implication of Theorem 5.1: Let $\alpha \geq \frac{1}{2}$. We want to find a cohesive pattern $G = (V, E)$. By Theorem 5.1, there exists a node $v \in V$ such that $G - v$ is α-dense and connected. Since the subspace cohesion constraint is anti-monotone by definition, pattern $G - v$ is a cohesive pattern as well. If we apply this top-down strategy recursively, we will end up with a cohesive edge. The set of all cohesive edges is exactly the input to *CoPaM*. Within the expand-by-one phase, current cohesive patterns are expanded by neighboring nodes. Therefore, the expand-by-one phase finds all cohesive patterns for $\alpha \geq \frac{1}{2}$.

Next, we show that indeed for $\alpha < \frac{1}{2}$ a different algorithmic strategy is necessary.

Proposition 5.3 *The connectivity and α density constraints together are **not** loose anti-monotone, if $\alpha < \frac{1}{2}$.*

Proof (by counter example). Figure 5.2 on page 42 shows a cohesive pattern for $\alpha = 0.41$. The only 0.41-removable node is c. Since c is a bridge node, its removal disconnects the graph, and therefore the connectivity constraint is violated. Therefore, the connectivity and density constraint together are not loose anti-monotone.□

5.4.3 Correctness of *CoPaM* for $\frac{1}{3} \leq \alpha < \frac{1}{2}$

In the last section, it was shown that the *CP* constraint is not loose anti-monotone for $\frac{1}{2} < \alpha$. We still can guarantee the correctness of *CoPaM* by applying the second phase (merging phase) as shown in the following. After introducing on additional definition, several lemmas are provided. This section is concluded with the proofs of the decomposition and completeness proof.

Definition 5.10 (Core Component, Leaf (Core Component)) *Let $G = (V, E, D)$ be a cohesive pattern and $\emptyset \neq CN(G) \subset V$ be the set of α-critical nodes in G. The connected components in $G - CN(G)$ are called **core components**. A core component which is connected to only one α-critical node is called **leaf core component** or **leaf** for short.*

An example of a graph containing leaf core components and core components can be found in Figure 5.4. The graphs contains four leafs, L_1 to L_4.

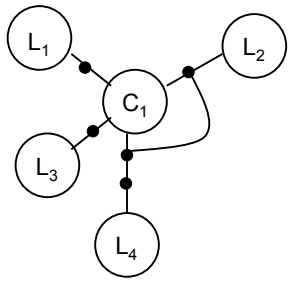


Figure 5.4: Illustration of Definition 5.10.
L_1, L_2, L_3 and L_4 are leaf core components; C_1 is a core component.

Lemmas

An illustration of the following lemma can be found in Figure 5.5.

Lemma 5.2 *Let G be a graph and let B be a subset of the bridge nodes in G. If $|B| > 0$ then G contains at least two connected components which are each connected to only one node in B.*

Proof Induction basis: If B contains only one node, then after the removal of this node, the graph results in at least two connected components which were connected to exactly one node in B, see left graph in Figure 5.5.
Induction step: Suppose B contains more than one node. After the removal on arbitrary node $b \in B$, the remaining graph consists of at least two connected components $CC_1, \ldots CC_i$, $i \geq 2$, see right graph in Figure 5.4. If CC_1 is connected to only one bridge node, then we stop. If CC_1, is connected to several bridge nodes, we apply the induction hypothesis to $CC_1 + b$. This results in at least two connected components which are connected to exactly one node in $B \setminus \{b\}$. However, one of these components might be b itself. Since $j \geq 2$, we use the same procedure for CC_2. Therefore, in total there are at least two connected components which are connected to exactly one node in B. □

Lemma 5.3 *If G contains at least one α-critical node, then G contains at least two leaf core components.*

Proof The α-critical nodes in G are a subset of all bridge nodes in G. Using Lemma 5.2, G contains at least two leaf core components.□

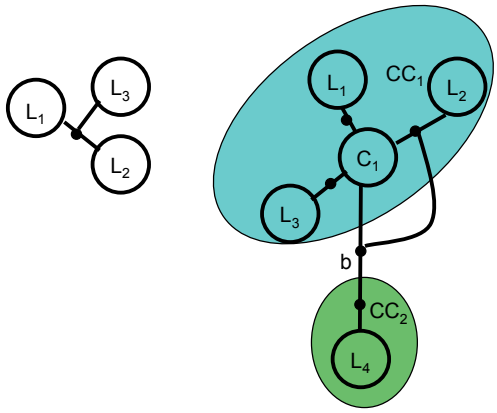


Figure 5.5: Illustration of Lemma 5.2.
The left graph contains only one bridge node. The right graph contains five bridge nodes. When choosing bridge node b, this results in the two highlighted components, namely CC_1 and CC_2.

The previous Lemma is very critical for the following proofs. Based on it, we know that any cohesive pattern contains at least two leaf core components.

Lemma 5.4 *The degree of any node in a leaf core component is at least* $\lceil \alpha(|V|-1) \rceil$.

Proof Let G be a cohesive pattern, which contains a leaf core component $L_1 = (V_1, E_1, D_1)$. Assume there exists a node $v \in V_1$, such that $deg(v) < \lceil \alpha(|V|-1) \rceil$. In the first part of the proof of Proposition 5.2, it is shown that every node with degree less than $\lceil \alpha(|V|-1) \rceil$ is α-removable. The removal of v either disconnects G or it does not:

- **Removal of v disconnects G:** This implies v is an α-critical node. Contradiction, core components cannot contain α-critical nodes.
- **Removal of v does not disconnect G:** Contradiction to the definition of core component - a core component cannot contain α-removable nodes.

Therefore, the degree of each node in L_1 is at least $\lceil \alpha(|V|-1) \rceil$.□

Lemma 5.5 *Let $G = (V, E, D)$ be an α-critical cohesive pattern. The size of a leaf is at least* $\lceil \alpha(|V|-1) \rceil$.

Proof By Lemma 5.4, it is known that each node in a leaf has a degree of at least $\lceil \alpha(|V|-1) \rceil$. However, each node in a leaf core component is connected to at most one α-critical node. In order to fulfill this degree requirement, the size of the component has to be at least $\lceil \alpha(|V|-1) \rceil$.□

Lemma 5.6 *Let $\frac{1}{3} \leq \alpha < \frac{1}{2}$ and $G = (V, E, D)$ be a cohesive pattern. The removal of an α-critical node from G disconnects G into exactly two leafs.*

Proof Let n be the number of nodes in G. Assume G contains three core components G_1, G_2 and G_3. By Lemma 5.3, there exist at least two leaf core components, say G_1 and G_2, and at least one α-critical node. By Lemma 5.5, both G_1 and G_2 contain more than $\alpha(n-1) \geq \frac{n-1}{3}$ nodes. In total, G_1 and G_2 contain more than $2(\frac{n-1}{3})$ nodes.

Thus, the size of $G - G_1 - G_2$ is less than $n - 2(\frac{n-1}{3}) = \frac{n+2}{3}$. Therefore, less than $\frac{n+2}{3}$ nodes need to cover at least one additional core component G_3 (and at least one α-critical node). Every node in G_3 must be connected to more than $\frac{n-1}{3}$ nodes - however, these nodes cannot be in G_1 or G_2. Therefore, the size of G_3 is at least $\frac{n+2}{3}$. However, there remain less than $\frac{n+2}{3}$ nodes upon removal G_1 and G_2 from G.

This is a contradiction, therefore G cannot contain more than two core components. Since by Lemma 5.3, G contains at least two leaf core components, these two core components are leaf core components. However, G can contain two leaf core components as the example in Figure 5.2 shows.□

Lemma 5.7 *Let $L_1 = (V_1, E_1, D)$ and $L_2 = (V_2, E_2, D)$ be the two leafs in an α-critical cohesive pattern $G = (V, E, D)$ and let $\alpha \geq \frac{1}{3}$, then $|V_1| < 2|V_2|$.*

Proof By Lemma 5.5,

$$\alpha(|V|-1) < |V_2| \tag{5.1}$$

Assume, $2|V_2| \leq |V_1|$. Therefore,

$$3\alpha(|V|-1) < |V_1| + |V_2|.$$

We know that $|V_1| + |V_2| = |V| - |CN(G)|$, therefore,

$$3\alpha(|V|-1) < |V| - |CN(G)|.$$

Then $\alpha < \frac{|V|-|CN(G)|}{3(|V|-1)}$. Now since $CN(G) \geq 1$ we get $\alpha < \frac{1}{3}$. This contradicts with $\alpha \geq \frac{1}{3}$. Therefore, $2|V_2| > |V_1|$.□

Lemma 5.8 *Let $G_1 = (V_1, E_1, D)$ and $G_2 = (V_2, E_2, D)$ be the two leafs in an α-critical cohesive pattern $G = (V, E, D)$ and let $\alpha \geq \frac{1}{3}$, then G_1 and G_2 are $\frac{1}{2}$-dense.*

Proof Assume G_1 is not $\frac{1}{2}$-dense. This means there exists a node $v \in V_1$ which is connected to less than $\frac{|V_1|-1}{2}$ nodes within G_1 and to at most one α-critical node. Therefore,

$$deg(v) < \frac{|V_1| - 1}{2} + 1 = \frac{|V_1| + 1}{2}.$$

Since v is not an α-critical node, we also know that $deg(v) > \alpha(|V| - 1)$. Since $\alpha \geq \frac{1}{3}$ and $CN(G) \geq 1$,

$$deg(v) > \alpha(|V| - 1) \geq \frac{1}{3}(|V_1| + |V_2| + CN(G) - 1) \geq \frac{1}{3}(|V_1| + |V_2|).$$

Combining the two above inequalities results in

$$\frac{1}{3}(|V_1| + |V_2|) < deg(v) < \frac{|V_1| + 1}{2} \tag{5.2}$$

We distinguish between $|V_1|$ being odd and even.

- $|V_1|$ **is odd:** Since degree of v must be integer,

$$deg(v) < \left\lceil \frac{|V_1| + 1}{2} \right\rceil \Rightarrow deg(v) \leq \frac{|V_1| - 1}{2} \tag{5.3}$$

  By Lemma 5.7, we have $\frac{|V_2|}{|V_1|} > \frac{1}{2}$. Since $|V_2|$ is an integer, we get

$$|V_2| \geq \frac{|V_1| + 1}{2}.$$

  Combining this result with (5.2) we get

$$\begin{aligned} \lceil deg(v) \rceil &\geq \left\lceil \frac{1}{3}(|V_1| + |V_2|) \right\rceil \\ &\geq \left\lceil \frac{1}{3}\left(|V_1| + \frac{|V_1| + 1}{2}\right) \right\rceil \\ &= \left\lceil \frac{|V_1|}{2} + \frac{1}{6} \right\rceil = \frac{|V_1| + 1}{2} \end{aligned} \tag{5.4}$$

  (5.3) and (5.4) are contradicting each other, therefore G_1 is $\frac{1}{2}$-dense, if $|V_1|$ is odd.

- $|V_1|$ **is even:**

$$deg(v) < \frac{|V_1|+1}{2} \Rightarrow deg(v) \leq \frac{|V_1|}{2} \tag{5.5}$$

By Lemma 5.7, we have

$$\frac{|V_2|}{|V_1|} > \frac{1}{2}.$$

Thus,

$$|V_2| \geq \frac{|V_1|+2}{2}.$$

From (5.5) we get,

$$\begin{aligned} deg(v) &\geq \left\lceil \frac{1}{3}(|V_1|+|V_2|) \right\rceil \\ &\geq \left\lceil \frac{1}{3}\left(|V_1| + \frac{|V_1|+2}{2}\right) \right\rceil = \frac{|V_1|+2}{2} \end{aligned} \tag{5.6}$$

Since (5.5) and (5.6) contradict each other, G_1 is $\frac{1}{2}$-dense, if $|V_1|$ is even.

Thus, G_1 is α-dense. Similar proof for G_2.□

Lemma 5.9 *Let $G = (V, E, D)$ be an α-critical cohesive pattern containing a set of α-critical nodes $CN(G)$. Let $c_1 \in CN(G)$ and $c_2 \in CN(G)$ be the α-critical nodes that the two leaf core components $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ respectively are connected to. Let also $G_1^* = (V_1^*, E_1^*) = G_1 + c_1$ and $G_2^* = (V_2^*, E_2^*) = G_2 + c_2$, then G_1^* and G_2^* are both α-dense.*

Proof For every node $v \in V_1$, by Lemma 5.4 $deg(v) > \alpha(|V|-1)$. Therefore,

$$|E_1^*| > \frac{|V_1|\alpha(|V|-1)}{2}.$$

Calculating the density of G_1^*, we get

$$\begin{aligned} d(G_1^*) &= \frac{2|E_1^*|}{|V_1|(|V_1|+1)} \\ &> \frac{2|V_1|\alpha(|V|-1)}{2(|V_1|(|V_1|+1))} \\ &= \frac{\alpha(|V_1|+|V_2|+CN(G)-1)}{|V_1|+1} > \alpha, \end{aligned} \tag{5.7}$$

since $|V| = |V_1| + |V_2| + |CN(G)|$. Therefore, G_1^* is α-dense. Analogous proof for G_2^*.□

Decomposition Theorem and Completeness Proof

Theorem 5.2 (Decomposition Theorem - first part) *Let G be an α-critical cohesive pattern containing the α-critical nodes $CN(G)$ and two leaf core components G_1 and G_2. There exists a non-disjoint partitioning of $CN(G)$ into C_1 and C_2, such that $C_1 \cup C_2 = CN(G)$, $|C_1 \cap C_2| \geq 1$ and $G_1 \cup C_1$ and $G_2 \cup C_2$ are α-dense and overlap.*

Proof Note that $CN(G)$ is a simple path. This is due to the fact that G_1 and G_2 are leaf core components. By definition, leaf core components can be connected to only one node in the set of α-critical nodes. Therefore, $CN(G)$ is a simple path.

By Lemma 5.9, we know G_1^* and G_2^* are α-dense. Let $G_1' = (V_1', E_1') := G_1 \cup C_1$ and $G_2' = (V_2', E_2') := G_2 \cup C_2$ denote the graphs resulting by adding one connected α-critical node at a time (from $CN(G)$) to G_1^* and G_2^* until the resulting graphs cannot be expanded any further.

Assume G_1' and G_2' do not overlap, *i.e.*,

- $|E_1'| + 1 < \frac{\alpha|V_1'|(|V_1'|+1)}{2}$ and
- $|E_2'| + 1 < \frac{\alpha|V_2'|(|V_2'|+1)}{2}$

Furthermore, there are $T \geq 1$ α-critical nodes not absorbed by G_1' or G_2'.

Note that $|E| = |E_1'| + |E_2'| + T - 1$, $|V| = |V_1'| + |V_2'| + T$. Let K be defined as $K = |V_1'| + |V_2'|$.

The density of G is calculated as follows:

$$\begin{aligned} d(G) &= \frac{2|E|}{|V||V-1|} \\ &= \frac{2(|E_1'| + |E_2'| + T - 1)}{(K+T)(K+T-1)} \\ &< \frac{2(\frac{\alpha|V_1'|(|V_1'|+1)}{2} + \frac{\alpha|V_2'|(|V_2'|+1)}{2} + T - 1)}{(K+T)(K+T-1)} \end{aligned}$$

$$= \alpha \left[\frac{|V_1'|(|V_1'|+1) + |V_2'|(|V_2'|+1) + \frac{2T-2}{\alpha}}{|V_1'|^2 + |V_2'|^2 + 2(|V_1'||V_2'|) + (2T-1)(|V_1'| + |V_2'|) + T^2 - T} \right] \tag{5.8}$$

Since $T \geq 1$, $|V_1'| \geq 4$ and $|V_2'| \geq 4$ (Lemma 5.10), therefore $K \geq 8$ which results in

$$\begin{aligned} |V_1'|(|V_1'|+1) &< |V_1'|(K+T-1) \\ |V_2'|(|V_2'|+1) &< |V_2'|(K+T-1) \end{aligned} \tag{5.9}$$

Note that

$$\frac{2(T-1)}{\alpha} < T(K+T-1)$$

because $\alpha \geq \frac{1}{3}$ and

$$\begin{aligned} \frac{2}{3}(T-1) &< T(8+T-1) \\ \Leftrightarrow \frac{2}{3} &< T\left(1+\frac{8}{T-1}\right) \end{aligned} \tag{5.10}$$

Since $T \geq 1$, inequality (5.10) is fulfilled.

Since the nominator in (5.8) is smaller than the denominator (5.10), the density of G is smaller than α which is a contradiction. Therefore, $G'_1 = G_1 \cup C_1$ and $G'_2 = G_2 \cup C_2$ overlap.□

Theorem 5.3 (Decomposition Theorem - second part) *Let G, $G_1 \cup C_1$ and $G_2 \cup C_2$ be defined as in the first part of the Decomposition Theorem.*
There exists G_1^{merge} and G_2^{merge}, $G_1^{merge} \subseteq (G_1 \cup C_1)$ and $G_2^{merge} \subseteq (G_2 \cup C_2)$, such that $|G_1^{merge} \cap G_2^{merge}| = 1$. Furthermore, G_1^{merge} and G_2^{merge} are expanded-by-one cohesive patterns and merging candidates.

Proof Let $G_1 \cup C_1$ and $G_2 \cup C_2$ be as defined in the first part of the Decomposition Theorem. Clearly, we can find a connected subgraph G_1^{merge} of $G_1 \cup C_1$ such that is overlaps with $G_2^{merge} := G_2 \cup C_2$ by *exactly one* node.
Recall that G_1 and G_2 are leaf core components. By Lemma 5.8, G_1 and G_2 are $\frac{1}{2}$-dense. Therefore, by Theorem 5.1, they are expanded-by-one cohesive patterns. Since G_1^{merge} and G_2^{merge} are generated from G_1 and G_2 by adding one connected node at a time and they are α-dense (by the first part of the theorem) - they are also expanded-by-one cohesive patterns.

It remains to show that G_1^{merge} and G_2^{merge} are merging candidates. Recall, a merging candidate is a $\frac{1}{2}$-dense graph together with a simple path P. However, the α-critical nodes form a simple path - see proof of first part of the Decomposition Theorem. Furthermore, G_1 and G_2 are $\frac{1}{2}$-dense (Lemma 5.8). Therefore, the definition of G_1^{merge} and G_2^{merge} follows the definition of merging candidates.□

The following lemma is not necessary for the proof of the completeness theorem, but it is necessary to guarantee the correctness of the definition of merging candidates. In the definition of merging candidates, we state that the size of the $\frac{1}{2}$-dense graph within the merging candidate, is at least 4. In the following lemma, we show that indeed there cannot exist smaller merging candidates.

Lemma 5.10 *The size of a merging candidate which can lead to an α-critical graph is at least 5.*

Proof The smallest α-critical graph contains a single α-critical node $c \in CN(G)$, $deg(c) \geq 2$. For c to be α-critical, its degree is the lowest in the whole graph. Since we want to get the smallest possible graph, $deg(c) = 2$. Therefore, the degrees of the nodes in the leaf core components G_1 and G_2 are at least 3. However, only one node per leaf core component can be connected to c, therefore, G_1 and G_2 contain at least 4 nodes. This means that the size of the merging candidates $G_1^{merge} = G_1 + c$ and $G_2^{merge} = G_2 + c$ is at least 5 and the size of G_1 and G_2 at least 4 - as stated in the definition.□

Note that node c in the proof above is actually not an α-critical node for any α. However, the derived lower bounds turn out to be very useful for many proofs, although they are not tight.

In the following, we prove the completeness of *CoPaM*. We want to show that for $\alpha \geq \frac{1}{3}$ any cohesive pattern is either expanded-by-one, *i.e.*, it will be found in the expand-by-one phase or it can be decomposed into two merging candidates and an expand-by-one part. To clarify, what we mean by *expand-by-one part*, we refer to Algorithm 3, Line 16. Here, we call the expand by one phase a second time expanding α-critical cohesive patterns by one neighboring node at a time. These neighboring nodes, which lead to a cohesive pattern, are called *expand-by-one part*.

Theorem 5.4 (Completeness Theorem) *If $\frac{1}{3} \leq \alpha$, a cohesive pattern G is either expanded-by-one or it can be decomposed into an expand-by-one part and two merging candidates which are expanded-by-one.*

Proof Let G be a cohesive pattern of size n. If G is expanded-by-one, we use the same argument as in Theorem 5.1. Otherwise, let $G_n = G$ be a cohesive pattern which is not expanded-by-one. Let v_n be an α-removable node in G_n which is not a bridge node, *i.e.*, $G - v_n = G_{n-1}$ is a cohesive pattern. We apply this strategy recursively until G_j, $j \leq n$, contains an α-critical node. This strategy corresponds to the second call of the expand-by-one phase (Line 16 of Algorithm 3).

By Lemma 5.6, G_j contains exactly two leaf core components G_1 and G_2 and a set of α-critical nodes $CN(G)$, *i.e.*, $G_j = G_1 \cup CN(G) \cup G_2$. By the Decomposition Theorem (first and second part), we know that there exists a non-disjoint partition, C_1 and C_2, $C_1 \cup C_2 = CN(G)$, $|C_1 \cap C_2| = 1$, of nodes in $CN(G)$ such that $G_1 + C_1$ and $G_2 + C_2$ overlap and are expanded-by-one. Therefore, they are found in the expand-by-one phase. The pattern $G_1 + C_1$ and $G_2 + C_2$ are merging candidates and are merged into G_j in the merge phase.□

Algorithmic implications: The correctness proof is done in a top-down manner, *i.e.*, for any cohesive pattern, we show that it can be decomposed in the described three components, two merging candidates and an expand-by-one part. However, *CoPaM* finds in a bottom-up manner all merging candidates and expands them afterwards. Since *CoPaM* follows exactly the proof, we have a guarantee that it finds all maximal cohesive patterns. In Figure 5.2, the two merging candidates are $G_1 + c$ and $G_2 + c$. The component $CN(G)$ consists only of node c.

5.5 Experiments

We evaluate *CoPaM* on three real world datasets, one social network dataset and two biological datasets. As other graph pattern mining algorithms [62], [76], *CoPaM* was evaluated in terms of efficiency and scalability on synthetic datasets. All experiments were performed on a PC running Linux with a 1.86GHz CPU and 4 GB of main memory.

5.5.1 Social Network Data

One of the applications of *CoPaM* is the identification of collaboration groups. We used the co-author dataset from Chapter 4.5. Recall that the feature vectors represented the words occurring in the abstracts of the respective authors. In this chapter, the feature vectors are transformed into Boolean values. A *one* means that the corresponding word occurred at least one time in the abstract of an author. A *zero* means it never occurred. We chose the following subspace cohesion function s_{sn}:

$$s_{sn}(V, D, true) = \forall v \in V, d \in D : F_d(v) = true,$$

where $F_d(v)$ denotes the feature vector of node v in dimension d. This subspace cohesion function requires that all members of a community have all keywords in subspace D in common. Furthermore, the threshold for the minimum dimensionality was set to 16 and the density α was set to $\frac{1}{3}$. Since there is no gold standard to evaluate the results on social network data, we discuss some anecdotal evidence of the meaningfulness of the cohesive patterns discovered.

In total, *CoPaM* identified 59 collaboration groups of size 6 or larger. As anecdotal evidence of the meaningfulness of cohesive patterns we discuss the following examples:

1. Wei Wang, Philip Yu, Jiawei Han, Beng Ooi, Kian-Lee Tan, Hongjun Lu (density = 0.4)
2. Philip Yu, Jiawei Han, Charu Aggarwal, Laks Laskhamanan, Divesh Srivastava, H. Jagadish (density = 0.5)

Jiawei Han and Philip S. Yu are part of both patterns. Depending on the topic they have collaborated with different researchers. According to the first pattern they worked with Wei Wang, Beng Ooi, Kian-Lee Tan and Hongjun Lu on statistical methods (some of the identified subspace dimensions are *skew, mixture* and *uniform*). According to the second pattern, they worked with Charu Aggarwal, Laks Lakshmanan, Divesh Srivastava and H. Jagadish on hierarchical document mining (*document, feature* and *hierarchical*). We also identified the cohesive pattern of size 18 which can be found in Figure 5.6. This pattern corresponds to the VLDB paper with the title *The Propel Distributed Services Platform*.

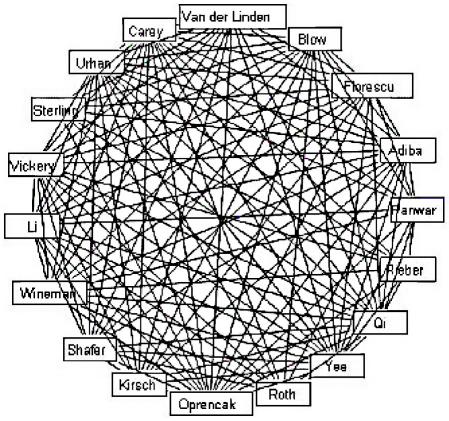


Figure 5.6: Largest cohesive pattern from social network dataset.

5.5.2 Biological Data

The two datasets under consideration are human and yeast. In both datasets the nodes correspond to genes, the edges to interactions (protein-protein and genetic interactions) and the feature vectors to gene expression data. It has been argued, (*e.g.*, [20, 70, 36]) that the combined analysis of these two data types for the identification of modules is more promising than the individual analysis. Indeed, the identified maximal cohesive patterns have a specific biological meaning, namely modules, as we will show in the following.

Human Dataset ($H.sapiens$**)**: The interaction network (graph data) was extracted from the BioGRID database [19], which integrates both protein-protein and genetic interactions from multiple publicly available datasets. For the expression data (feature vectors), the comprehensive human tissue expression dataset [21] was used. As suggested by the authors, we retained only variably expressed

genes which showed at least 2-fold ratio variation from the mean in at least two experiments. The final dataset contains 3,628 nodes connected by 8,924 edges and 115 dimensions.

Yeast Dataset ($S.cerevisiae$): The interaction network (graph) was also retrieved from the BioGRID database [19]. The gene expression data (feature vectors) was acquired from Huges et al. [22], which contains fold changes of genes in 300 cDNA experiments. The final dataset contains 1,043 genes with 2,664 interactions and 300 dimensional feature vectors.

Due to the absence of comprehensive module annotations, a common method for evaluating module inference algorithms is testing for statistically significant over-represented biological process gene ontology (GO) terms in the group of interest. The GoMiner tool[1] is used for testing whether the maximal cohesive patterns are enriched with GO terms with P-values below a threshold of 0.01. The following three metrics are used to evaluate the quality of the results:

1. The *enrichment (precision)* is computed as the percentage of cohesive patterns that are enriched with at least one GO term.

2. The *coverage (recall)* is defined as the number of GO terms associated with an enriched cohesive pattern divided by the number of all GO terms in the dataset.

3. The *F-Measure* captures the trade-off between precision and recall. Given enrichment E and coverage C, it is computed as $F = \frac{2EC}{E+C}$.

The subspace cohesion function, s_{bio}, is defined as

$$s_{bio}(V, D, \theta_s)$$

$$= \forall d \in D : |\max\{F_d(v), v \in V\} - \min\{F_d(v), v \in V\}| \leq \theta_s.$$

The function $F_d(v)$ denotes the value of v's feature vector in dimension d. This cohesion function requires that the expression values (or fold changes) of all genes (nodes) in a pattern induced by V are within a range of θ_s in all experiments (dimensions) D. Note that if $F_d(v)$ refers to missing data, s_{bio} is false.

To assess the quality of *CoPaM* of the biological dataset, we compared it to two related state-of-the-art algorithms that operate on both graph data and feature vectors, MATISSE [70] and Co-clustering [36]. For the comparison partners, we used the recommended parameter settings. For

[1] http://discover.nci.nih.gov/gominer/

Table 5.2: Quality assessment on the human dataset

Algorithm	Coverage	Enrichment	F-measure
Co-Clustering	**0.65**	0.79	0.71
MATISSE	0.38	0.93	0.54
CoPaM	0.64	**0.96**	**0.77**

Table 5.3: Quality assessment on the yeast dataset

Algorithm	Coverage	Enrichment	F-measure
Co-Clustering	0.42	0.71	0.53
MATISSE	0.17	0.94	0.29
CoPaM	**0.59**	**0.95**	**0.73**

CoPaM, the density threshold was set to 0.65, based on the density distribution of known modules[2]. For yeast, the fold-change range threshold (θ_s) was set to 1.25 and the minimum dimensionality to 140, which is also derived from the true modules. For the human dataset, we used a more relaxed parameter setting due to the high amount of noise and missing values in the human gene expression data. We used 1.4 for the fold-change range threshold (θ_s) and 10 for the minimum dimensionality.

Results

Table 5.2 and Table 5.3 show the results from the human and yeast datasets respectively. The best score for each metrics is printed in bold font. In both datasets, MATISSE and *CoPaM* consistently yield enrichment over 90%. However, coverage-wise we see that only Co-Clustering and *CoPaM* yield scores over 60% in the human dataset and *CoPaM* achieves the top coverage by a large margin in the yeast dataset. MATISSE outputs only the statistically significant patterns, hence it achieves high enrichment. However, it performs poorly in terms of coverage. On the other hand, Co-Clustering forces every node into a pattern, hence it yields high coverage at the cost of poor enrichment. Although *CoPaM* does not force every node to belong to a pattern, it achieves comparable, if not better, coverage, due to its completeness. This means that *CoPaM* is able to find patterns of a larger range of functionalities without sacrificing quality. This is supported by the F-measure, in which *CoPaM* performs the best.

CoPaM finds connected, dense and homogeneous patterns. It allows overlap and does not force every node into a pattern. None of the comparison partners address all of these issues simultaneously and we argue this is the main reason for the superiority of *CoPaM*. The runtime was 8 seconds for

[2]http://www.yeastgenome.org

Table 5.4: Runtime of synthetic datasets.

Number of nodes	746	1,492	2,238	2,984	3,730
Runtime in seconds	50	111	306	421	560
Number of cohesive patterns	667	1,347	2,037	2,723	3,419

the human dataset and 18 seconds for the yeast dataset.

5.5.3 Synthetic Data

In this section, we analyze the runtime of *CoPaM* on synthetic datasets. The synthetic FVNs are based on the social network dataset described in Section 5.5.1. We took the largest component after the removal of non-cohesive edges and made several copies of it, connecting the components randomly with cohesive edges. These graphs have the property that they are connected after the preprocessing phase. We chose the following parameter settings: $\alpha = \frac{1}{3}$, s is the same as for the social network dataset, minimum dimensionality is set to 20.

We generated graphs with the sizes of 746, 1492, 2238, 2984, 3730. The total runtimes and number of maximal cohesive patterns can be found in Table 5.4. The size of the largest pattern is 19. The runtime ranges from 50 seconds for the graph of size 746 to 560 seconds for the graph of size 3730. The number of maximal cohesive patterns is between 667 and 3419. Let us now have a closer look at the different levels of these runtimes. Recall that a level in *CoPaM* corresponds to the step of generating cohesive patterns of size n based on the ones of size n-1. For example, on level 7, where the time for generating cohesive patterns of size 7 based on the ones of size 6 is measured, we recognize a linear trend, see Figure 5.7. The data points in this figure correspond to the runtime for the different synthetically generated graphs. The runtime for generating 43,978 patterns in the graph of size 746 is 5 seconds and increases linearly with the number of cohesive patterns, up to 26 seconds for 238,489 patterns in the graph of size 3,730.

5.5.4 Comparison to Baseline Approaches

We compare the number of patterns produced by the baseline algorithms introduced in 5.3.1 (called Baseline 1 and 2) versus *CoPaM*. All algorithms are output-sensitive algorithms in each level (=size of patterns), therefore the runtime is reflected by the number of patterns. That was also shown experimentally in the previous subsection. The dataset is the same as the one described in Section 5.5.1 with the stated parameters. For the generation of patterns up to a size of 5 (a larger number caused a memory overflow), Baseline 1 generated 70 times more patterns than *CoPaM* and Baseline

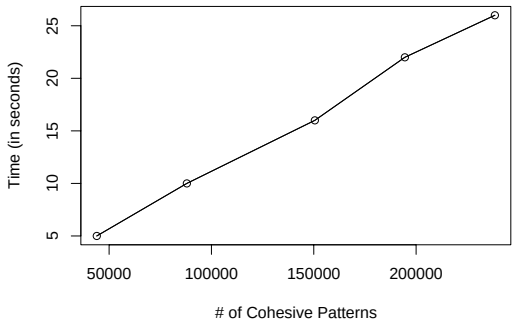


Figure 5.7: Runtime for generating cohesive patterns.
Time it takes to generate cohesive patterns of size 7 based on cohesive patterns of size 6.

2 eleven times.

5.6 Extensions of *CoPaM*

CoPaM can be extended in various ways. In this section, we discuss two extensions: a parallelized version of *CoPaM* and additional constraints for cohesive patterns.

5.6.1 Parallelization of *CoPaM*

In some applications, memory and/or speed of *CoPaM* might be an issue due to the number of patterns mined by *CoPaM*. For these cases, we propose a parallelized version of *CoPaM*, called *paraCoPaM*. In the following, we explain its differences to *CoPaM*. The pseudo code of *paraCoPaM* can be found in Algorithm 6. The algorithm does not require shared memory.

The general idea of *paraCoPaM* is to divide the input FVN into several partitions and mine the cohesive patterns in these partitions in parallel, see also Figure 5.8. In this figure, the input FVN is partitioned into G_1 and G_2. Concretely, the input to *paraCoPaM* is the same as to *CoPaM* with one additional parameter, k, reflecting the number of desired processes. In line 3 of Algorithm 6, all non-cohesive edges are removed from $\mathcal{G}$. Our goal is to mine the patterns in $\mathcal{G}$ in parallel. However, it has been previously reported [4] that the size distribution of the connected components in biological (and social) networks is highly skewed, *i.e.*, these networks often contain one giant connected component along with many other small ones. Therefore, in order to parallelize the computation for the FVN and in particular for the large component(s), we partition the nodes of $\mathcal{G}$ into k balanced partitions

$P_1, \ldots P_k$ and analyze these partitions independently. The partition is achieved by applying the k-way normalized cut [75], a graph cut algorithm which guarantees balanced components. Dhillon et al. [13] propose a very fast approximation algorithm, called graclus, to solve the k-way normalized cut problem.

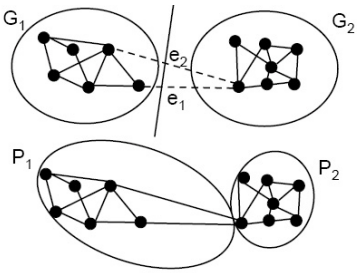


Figure 5.8: Illustration of *paraCoPaM*.
The top figure shows the result of applying the 2-way cut to a graph $\mathcal{G}$. The bottom figure shows the assignment of the edges to the two processes P_1 and P_2 in the initialization phase.

These k smaller partitions are independently mined in parallel. In order to still guarantee correctness, the first phase of *CoPaM* has to be adapted slightly. When expanding cohesive patterns, not only all neighboring nodes *within the current partition* have to be considered, but also connected nodes from *neighboring partitions*. After the first phase is finished, the identified cohesive patterns and merging candidates are combined. If α is less than $\frac{1}{3}$, the second phase has to be applied to the merging candidates. Finally, as in the case of *CoPaM*, a maximality check is required.

We explain this extension now based on the following example. As in Figure 5.8, assume $k = 2$. Therefore, the normalized cut procedure returns two partitions, namely P_1 and P_2. Partition P_1 is analyzed by $Proc_1$ and P_2 by $Proc_2$. The seeds induced by the edges in the cut (e_1 and e_2) will be added to the machine which handles fewer edges ($Proc_1$). For the extension of the nodes in P_1, we need to consider not only the nodes within P_1 but also the nodes in P_2. This way, no cohesive pattern will be missed during the first phase of *paraCoPaM*. The second phase of *paraCoPaM* stays the same as in *CoPaM*.

5.6.2 Additional Constraints on Cohesive Patterns

In some applications, additional constraints on the cohesive patterns are desirable. In the following, we introduce two constraints: one in which the input graph has edge weights and one in which

Algorithm 6 *paraCoPaM:* Parallelized Cohesive Pattern Miner

INPUT: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{A}), \alpha, s, \theta_s, \theta_{dim}$, k
OUTPUT: maximal cohesive patterns
PREPROCESSING: remove non-cohesive edges from $\mathcal{G}$
$P_1, \ldots P_k$:=NormalizedCut($\mathcal{G}$, k)
for all parallel (partition $P_i = (V_i, E_i)$ in $\mathcal{G}$) **do**
  // FIRST PHASE: EXPAND-BY-ONE
  $currCohesivePatterns_i \leftarrow \emptyset$
  $mergingCand_i \leftarrow \emptyset$
  for all (edges $e = \{v_1, v_2\} \in E_i$) **do**
    $G_e \leftarrow (\{v_1, v_2\}, \{\{v_1, v_2\}\}, D)$
    $currCohesivePatterns_i$.add(G_e)
  end for
  $currCohesivePatterns_i, mergingCand_i \leftarrow$ Expand-by-one($currCohesivePatterns_i$)
end for
// COMBINE RESULTS
$currCohesivePatterns = \cup_i currCohesivePatterns_i$
$mergingCand = \cup_i mergingCand_i$
if ($\alpha < \frac{1}{2}$) **then**
  // SECOND PHASE: MERGE
  $mergedPatterns \leftarrow$ merge($mergingCand$)
  $currCohesivePatterns$.add(Expand-by-one($mergedPatterns$))
end if
MAXIMALITY CHECK: remove non-maximal cohesive patterns from $currCohesivePatterns$
return $currCohesivePatterns$

several input graphs are provided.

Weighted Cohesive Patterns

In the following, we assume that the FVN contains edge weights. In a social network, edge weights could be values reflecting the trust between two friends or the number of exchanged instant messages. Formally, a weighted graph is defined as follows:

Definition 5.11 (Edge weight function, weighted FVN) *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F})$ be a FVN. Let*

$$w : E \rightarrow \mathbb{R}$$

*be an function, called **edge weight function**, which assigns to every edge $e \in E$ a weight $w(e) \in \mathbb{R}$, then $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F}, w)$ is called **weighted FVN**.*

A user might be interested in mining all cohesive patterns whose *weight* exceeds a certain threshold. There are several alternative definitions for the weight of a cohesive pattern. Two of them are introduced in the following:

Definition 5.12 (Weight of Cohesive Pattern) *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F}, w)$ be a weighted FVN and $G = (V, E, D)$ be a cohesive pattern.*

- *The **average weight** of G is defined as: $w_{avg}(G) = \frac{\sum_{e \in E} w(e)}{|E|}$.*
- *The **relative weight** of G is defined as: $w_{rel}(G) = \frac{\sum_{e \in E} w(e)}{\frac{|V|(|V|-1)}{2}}$.*

Based on this definition, a weighted cohesive pattern is introduced:

Definition 5.13 (Weighted Cohesive Pattern, Weight Constraint) *Let $0 \leq \theta_w \leq 1$ be a weight threshold. A cohesive pattern G is called **weighted cohesive pattern** if $w_{avg}(G) > \theta_w$, respectively $w_{rel}(G) > \theta_w$. (Weight Constraint)*

Every weighted cohesive pattern is also a cohesive pattern. Therefore, for both weight definitions, *CoPaM* is able to find all weighted cohesive patterns efficiently by checking the weight constraint for cohesive patterns during the mining process. In case of the average weight and $w : E \rightarrow [0, 1]$, our loose anti-monotonicity proof for the simultaneous satisfaction of density and connectivity constraints for $\alpha \geq \frac{1}{2}$ (Theorem 5.1) can be easily extended as follows:

Theorem 5.5 *Simultaneous satisfaction of the connectivity and density constraints for weighted cohesive patterns (average weight) is loose anti-monotone for $\alpha \geq \frac{1}{2}$.*

Proof Let $G = (V, E, D)$ be connected and α-dense for $\alpha \geq \frac{1}{2}$. Assume the only α-removable nodes in G are bridge nodes. Let $b \in V$ be one of them. Then $G - b$ contains at least two connected components G_1 and G_2. There is at least one node in G_1, say v_1, whose removal does not disconnect G (application of Lemma 5.1 to $G_1 + v$). Since v_1 was not α-removable, $w_G(v_1) > \alpha(|V| - 1)$, where $w_G(v_1) = \sum_{\{v_1, u\} \in E} w(\{v_1, u\})$. Since $w(\{v_1, u\}) \leq 1$, G_1 contains more than $\alpha(|V| - 1) - 1 + 1 \geq \frac{1}{2}(|V| - 1)$ nodes. Using a similar argument for G_2 results in G having more than $|V|$ nodes, which is a contradiction. Therefore, there exists a node $v \in V$, such that $G - v$ is connected and α-dense.□

Cohesive Patterns across Multiple Feature Vector Networks

In social network analysis, we might have not only the friendship relationships between people - which could be based on their Facebook friends - but also their friends on their instant messenger. In order to mine several networks simultaneously, the user might be interested in several different

scenarios. He might want to examine the commonalities of networks in order to find out, for example, the *strength* of a friendship. In other scenarios, he might be interested in the differences between (sets of) networks.

5.7 Conclusion and Future Work

While most existing methods for analyzing network data use graph data only, in many applications, data have the form of FVNs. Recently, methods for mining such FVNs have emerged in several research communities. To mine patterns in such networks, we introduced the novel problem of mining cohesive patterns, which combines the concepts of dense subgraphs and of subspace clusters into a problem definition with important real life applications such as social network analysis and systems biology. The task is to find all maximal cohesive patterns, *i.e.*, dense and connected subgraphs with feature vectors that are homogeneous in a sufficiently large subspace. The proposed *CoPaM* algorithm makes the computationally hard problem tractable by simultaneously pruning the search space based on the three constraints (density constraint, subspace cohesion constraint and connectivity constraint) imposed by the definition of a cohesive pattern. We prove that *CoPaM* is complete for a density threshold above $\frac{1}{3}$. Our experiments on real life datasets show that *CoPaM* produces meaningful patterns. Our evaluation on synthetic data demonstrates the scalability of *CoPaM*.

We conclude by discussing several directions for future research. An interesting research direction is the simultaneous analysis of several FVNs. These FVNs might be observations of different time points or different types of relationships, such as friendships, phone call networks and exchanged instant messages. A further interesting direction is the tracking of changes in communities over time.

One problem we faced when evaluating *CoPaM* was the lack of publicly available social network datasets. Furthermore, to the best of our knowledge, a generator for FVNs does not exist. To simulate realistic FVNs is a very hard task, since feature vectors and network structure are dependent on each other and cannot be generated independently. This inspired our work on a simulator for FVNs which will be introduced in the next chapter.

Chapter 6

Simulation of Feature Vector Networks

The outbreaks of SARS (Severe Acute Respiratory Syndrome) in 2002 and the Swine flu (H1N1) in 2009 highlight the importance of epidemiological research. Epidemiologists could be interested in the simulation of the disease spread in networks. The network structure is particularly important, since influenza spreads between human beings through coughing, sneezing and touching [1]. However, not every individual is equally receptive to these viruses, and some people might have already been vaccinated. The receptiveness and immunity might be modeled by adding features to the persons in the social network, thus resulting in FVNs. Obtaining such real-world FVNs is very difficult for several reasons. One of the reason is the cost, another is the concern for privacy. To the best of our knowledge, existing graph generators can only generate networks, but not FVNs. Since in most cases feature vectors and network structure are dependent on one another, an independent generation of both data types does not deliver the desired result. This indicates a clear need for a simulator for FVNs.

In this chapter, we introduce a new methodology for simulating FVNs. In particular, we are able to simulate networks which are consistent with an estimated statistical model. The two statistical models under consideration are the previously introduced Latent Socio-Spatial Process (LSSP) model [47] and its extension, the n^*LSSP model, introduced in this chapter. The main goal of these models is to predict links in FVNs. The n^*LSSP model takes into account effects related to the network size using dampening functions and aids in scaling networks to an arbitrary size. Our experiments on several datasets show that the parameters in the n^*LSSP mode can indeed be estimated.

6.1 Introduction

Human-to-human infectious diseases such as Influenza or sexually transmitted diseases (STDs) have very high death toll. Given limited budget and resources, researchers also need to rely on theoretical studies, in which they simulate the spread of these diseases in order to identify the subset the people to vaccinate and/or educate. Observing real social networks is very expensive and often impossible for privacy reason. Therefore, researchers often make use of simulated networks. Unfortunately, existing models for simulating networks cannot generate feature vectors. Since an independent simulation of the network structure and feature vectors does not reflect realistic networks, there is a need for a simulation model for FVNs. There are several requirements on such a model which we discuss in the following.

6.1.1 Motivation

In the following, we use two examples, based on which we identify critical questions answered in this chapter. Consider a village of 20 people. It is very likely that everyone knows everyone, independent of their socio-economic status. However, in larger cities sociologists find that the socio-economic status and other features such as age, gender and education level have a high impact on the choice people make in terms of their friendships [71], [46]. Thus, the way in which friendship relationships are formed depends on the number of people in the population. The larger the population, the more selective people can be when forming friendships. One question, someone might ask now is, how to integrate the population size into a network model. As a second example, consider high schools. Further, let's assume we know the social network of two high schools; one with 500 students and the other with 1000. How can we simulate the social network of a school with 1500 students based on the statistical properties of the smaller networks?

The first contribution of this chapter is the n^*LSSP model, which is, to the best of our knowledge, the first statistical model for FVNs which tackles the issue of taking into account the population size. The n^*LSSP model is based on the recently introduced LSSP model and extends this model by integrating two dampening functions.

The underlying idea of the LSSP and n^*LSSP model is to project the features of people into a so-called *social space*, such that two individuals who are close in social space, are more likely to be connected than others who are not. This social space is modeled by a latent function whose parameters are estimated using Markov Chain Monte Carlo (MCMC).

We also introduce a framework for simulating FVNs based on the LSSP and n^*LSSP model.

The simulated networks have similar statistical properties to the original FVN/set of original FVNs. When simulating networks of a significantly different size than the input network, certain issues which we label as *constant link probability* and as *constant distance in social space* arise. The n^*LSSP model provides solutions for both issues.

6.1.2 Contributions

The main contributions that we present in this chapter are as follows:

- We introduce the first methodology for simulating FVNs.
- We propose the n^*LSSP model, an adaptation of the existing LSSP model. The n^*LSSP model takes into account effects related to the network size by introducing two dampening functions.
- We evaluate the n^*LSSP model on several synthetic datasets.
- We evaluate the methodology for simulation of FVNs.
- We discuss the impact of the model parameters on the link probabilities.

Overview

In Section 6.2, we discuss the existing literature in the area of statistical network modelling. Next, we review the LSSP model (Section 6.3), on which the n^*LSSP model, introduced in Section 6.4, is based. In the aforementioned two sections, we also propose two algorithms for simulating FVNs, called *simLSSP* and *simn*LSSP*. An evaluation of the n^*LSSP model can be found in Section 6.5. Section 6.6 provides a parameter sensitivity analysis. This chapter is concluded in Section 6.7.

6.2 Related Work

In this section, first some preliminaries are discussed. Next, state-of-the-art statistical models for FVNs are reviewed.

6.2.1 Preliminaries

Most of the models in this chapter focus on social networks. Accordingly, the term *actor* and *node* are used interchangeably. Recall, that $\boldsymbol{y}$ denotes an $n \times n$-dimensional symmetrical adjacency matrix

and $\boldsymbol{x}$ an $n \times d$-dimensional covariate matrix. The domain of $\boldsymbol{x}$ is denoted by $\mathcal{D}$. Capital letters are used to denote random variables and lower case letters to denote observations. Vectors and matrices are bolded. Therefore, the observed socio-matrix (denoted by $\boldsymbol{y}$) is a realization of the random variable $\boldsymbol{Y}$. Let $Y_{i,j}$ denote the ijth element of $\boldsymbol{Y}$. The probability associated with $Y_{i,j}$ is defined as:

$$Pr(Y_{i,j} = 1) = \pi_{i,j},$$

where $\pi_{i,j}$ is called **link probability**. Analogous, the probability for not having a link between i and j is defined as:

$$Pr(Y_{i,j} = 0) = 1 - \pi_{i,j}.$$

People tend to form friendships with others based on their respective features, such as age, gender or educational level. Often individuals tend to befriend similar people, which is also know by the term *social selection*, *e.g.*, Lazarsfeld et al. [46]. Therefore, most models in this chapter assume the edges, and therefore also the edge probabilities, to be independent conditional on the covariates, $\boldsymbol{x}$, and the model parameters, $\boldsymbol{\theta}$. Based on this, the link probability $\pi_{i,j}$ is a short form of the function $\pi(\boldsymbol{x_i}, \boldsymbol{x_j}, \boldsymbol{\theta})$. Furthermore, the probability of an observed network $\boldsymbol{y}$ is:

$$Pr(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{x}) = \prod_{i>j} Pr(Y_{i,j} = y_{i,j}|\boldsymbol{\theta}, \boldsymbol{x_i}, \boldsymbol{x_j}) =$$
$$\prod_{i>j} \pi_{i,j}^{y_{i,j}} (1 - \pi_{i,j})^{1-y_{i,j}}.$$

Modeling of Link Probabilities

Often, the $\pi_{i,j}$'s are not modeled directly. Instead a logistic transformation (inverse logit), $ilogit$, is used as link function [51]. The logistic transformation is defined as

$$\pi_{i,j} = ilogit(\eta_{i,j}) = \frac{exp(\eta_{i,j})}{1 + exp(\eta_{i,j})}.$$

Similar to $\pi_{i,j}$, the term η_{ij} is the short form for $\eta(\boldsymbol{x_i}, \boldsymbol{x_j}, \boldsymbol{\theta})$. In the models below, the probability $\pi_{i,j}$ is defined as $ilogit(\eta_{i,j})$. This transformation guarantees that the link probability $\pi_{i,j}$ is in $[0; 1]$. The relationship between η and $ilogit(\eta)$ is depicted in Figure 6.1.

Remark

Some of the models below cannot directly deal with FVNs, but require the (node-specific) features (*e.g.*, $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$) to be transformed into edge-specific features $\boldsymbol{x_{i,j}}$. A possible transformation

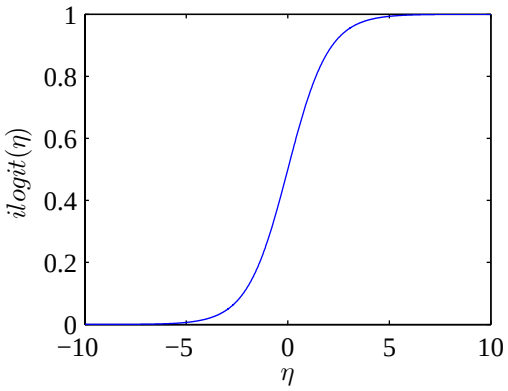


Figure 6.1: Logitistic transformation of η.

calculates the (component-wise) difference between the feature vectors, *i.e.*, $\boldsymbol{x_{i,j}} = |\boldsymbol{x_i}$ - $\boldsymbol{x_j}|$. In the following, with $\boldsymbol{x_{i,j}}$ we refer to this edge-specific feature vector.

6.2.2 Statistical Models for FVNs

In this thesis, we focus only on discriminative models. Discriminative models can be used to predict links and are of the form $Pr(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{x})$. We review the three dominant types of discriminative models. First, the Logistic Regression model is reviewed, afterwards the Latent Space Model and several of its extensions. The LSSP model will be explained in detail in Section 6.3.

Logistic Regression Model

A straight-forward statistical approach for link prediction in FVNs is the standard logistic regression model [51]. Logistic regression is a generalized linear model typically used for binary regression models. As before $\pi_{i,j} = \frac{exp(\eta_{i,j})}{1+exp(\eta_{i,j})}$, where

$$\eta_{i,j} = \boldsymbol{\beta'} \boldsymbol{x_{i,j}}.$$

The linear form of this model restricts the types of networks it is able to model. Furthermore, it does not take into account the $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ separately, but only in the form of $\boldsymbol{x_{i,j}}$.

Latent Space Models

The general idea of the latent space model is to introduce a latent space, which is called *social space*. Actors are projected into this social space, such that the closer two actors are in the social space, the more likely they are connected with each other. On the other hand, the further apart they are in social space, the less likely they are connected. The reason for introducing this social space is that sociologists observed that people tend to befriend similar people more likely than dissimilar people [46]. In Hoff et al. [38], a latent space model of the following form is introduced:

$$\eta_{i,j} = \beta_0 + \boldsymbol{\beta'} \boldsymbol{x_{i,j}} - \|\boldsymbol{z_i} - \boldsymbol{z_j}\|.$$

The $\beta_0 + \boldsymbol{\beta'} \boldsymbol{x_{i,j}}$ part of the equation reflects the impact of the covariates on the link probabilities. The $\boldsymbol{z_i}$'s are the latent positions of the actors in the social space. We can see, the closer two actors are, the more likely they are connected via an edge. The further apart they are, they less likely they are connected.

Handcock et al. [35] introduced an extension of the Latent Space Model for networks containing communities. In these models, the latent factors $\boldsymbol{z_i}$ are generated from a Multivariate Normal mixture model. More formally,

$$\boldsymbol{z_i} \sim \sum_{g=1}^{G} \lambda_g MVN_d(\boldsymbol{\mu_g}, \sigma_g^2 \boldsymbol{I_d}),$$

where λ_g is the probability of an actor belonging to the g-th group. These groups can be interpreted as communities. Therefore, this model can be used to identify the communities in networks and, using a Bayesian framework, it is able to discover the number of communities.

Heterogeneity in activity across nodes is another important feature of networks. In order to take this feature into account, Krivitsky et al. [45] extended the Latent Space Model by two random effects, denoted with δ_i and λ_j which model the sociality of people. The model looks as follows:

$$\eta_{i,j} = \beta_0 + \boldsymbol{\beta'} \boldsymbol{x_{i,j}} - \|\boldsymbol{z_i} - \boldsymbol{z_j}\| + \delta_i + \lambda_j.$$

The δ_i's and λ_i's are sender and receiver specific effects which model the heterogeneity in activity across nodes.

Discussion of Latent Space Models

If the z_i's are two-dimensional, their values can be used as coordinates. Therefore, the latent space aids in visualizing the actors in a meaningful way. The drawback of these models is that the number of parameters is linear in the number of actors. This often leads to overfitting, see Mitchell [54]. Last, these models cannot take into account the node-specific feature vectors, but only the edge-specific feature vectors.

To the best of our knowledge, all existing models focus on a single network. In this thesis, we introduce a new model called n^*LSSP model which can deal with effects related to the size of networks and provide with a single model for several networks (of various sizes). The n^*LSSP model is based on the LSSP model, which is introduced in the next section.

6.3 Latent Socio-Spatial Process Model

In this section, first the LSSP model is reviewed and afterwards applied to a real-world dataset. Next, we introduce *simLSSP*, a novel algorithm for simulating FVNs based on the LSSP model.

6.3.1 Model

In Linkletter [47] the **LSSP model** was introduced. The LSSP model is central to this chapter, since our methodology for simulating FVNs is based on it. Furthermore, it is extended in the next section.

Similar to the previously discussed models, the LSSP model is also based on the fact that similar people are more likely to be connected than dissimilar people. Similarity of people can be defined in various ways. Often similarity is modeled based on the similarity of the feature vectors of people. However, sometimes this leads to counter-intuitive cases. For example, consider a high school. Typically, high school students are friends with students of the same or similar grade. However, once they start dating, this changes, as can be seen in the real-world example below.

Therefore, the LSSP model determines the similarity of people by their distance in social space. The social space is modeled by a latent function that takes the covariates as input. The link probabilities are based on the distance between the two respective actors in this latent space. Furthermore, the LSSP model assumes the edges to be independent random variables given the covariate information and model parameters. In order to provide a complete model, some of the aforementioned probabilities are reviewed again. The LSSP model looks as follows:

$$Pr(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{x}) = \prod_{i>j} Pr(Y_{i,j} = y_{i,j}|\boldsymbol{\theta}, \boldsymbol{x_i}, \boldsymbol{x_j}) =$$

$$\prod_{i>j} \pi_{i,j}^{y_{i,j}} (1 - \pi_{i,j})^{1-y_{i,j}},$$

where

$$\pi_{i,j} = ilogit(\eta_{i,j})$$

and

$$\eta_{i,j} = \mu - |z(\boldsymbol{x_i}) - z(\boldsymbol{x_j})|.$$

This definition of $\eta_{i,j}$ is one of the critical ideas in the LSSP model. Recall that $\eta_{i,j}$ is the log-odds that actor i and j are connected with each other. This term consists of a constant μ and the absolute difference of the values of function z at the covariates $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$. The parameter μ is an upper bound for the probability that any two people are connected. The function $z : \mathcal{D} \rightarrow \mathbb{R}$ is latent and takes as input $\boldsymbol{x_i}$. It is used to project the covariate information of an actor into the latent social space; the value $z(\boldsymbol{x_i})$ is called *LSSP score* of $\boldsymbol{x_i}$. This is different from the latent space model, where $\boldsymbol{z_i}$ denoted a latent random variable which was independent of $\boldsymbol{x_i}$.

The term $\eta_{i,j}$ fulfills the properties we mentioned before: the larger the absolute distance in social space, the smaller the probability that two people are connected. The smaller the distance, the higher the probability that they are connected. Formally, this function z is defined as:

$$z(\boldsymbol{x_i}) = \sum_{r=1}^{m} \alpha_r k(\boldsymbol{x_i} - \boldsymbol{w_r}), \tag{6.1}$$

where k is a kernel function. An independent d-dimensional multivariate Gaussian kernel was chosen, thus:

$$k(\boldsymbol{x_i} - \boldsymbol{w_r}) = \prod_{l=1}^{d} \rho_l^{(w_{rl} - x_{il})^2}, \tag{6.2}$$

where $\mathcal{W} = \{\boldsymbol{w_1}, \ldots, \boldsymbol{w_m}\}$. Furthermore, $\boldsymbol{w_r} = (w_{r1}, \ldots, w_{rd})$, for $1 \leq r \leq m$, where d is the number of covariate dimensions. The $\boldsymbol{w_r}$, $1 \leq r \leq m$, are points generated based on a Latin Hyper Cube Design (LHD) [52]. They are also called *design points*, see below for details. The parameter $m = 10d$ denotes the number of design points.

The ρ_l's, $l = 1, \ldots d$, are parameters and need to be estimated. These parameters determine the impact of each dimension on the link probability. They are between 0 and 1. Values close to 0 show

a high impact of the dimension, whereas values close to 1 indicate a low impact. The rest of the parameters are the α_r's, $1 \leq r \leq m$. An α_r is the weight of its respective kernel. The importance of their corresponding kernel is reflected by the absolute value of α_r. Next, we review, how the parameter estimation is performed. In the following, we use the following short hand notations: $\boldsymbol{\rho} = (\rho_1, \dots \rho_d)$ and $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_m)$.

Parameter Estimation

Linkletter [47] suggests a Bayesian approach to estimate the parameter μ and the parameters $\boldsymbol{\alpha}$, $\boldsymbol{\rho}$, that are associated with the latent function z. In order to avoid large matrix inversions, they use a convolution model [49]. Furthermore, they discretize the convolution to substantially reduce the number of parameters. The previously mentioned set $\mathcal{W}$ denotes these discrete design points.

As stated before, the parameters which need to be estimated are: μ, $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$ which results in $1 + 10d + d = 1 + 11d$ parameters. Since a Bayesian approach was chosen, prior distributions need to be specified. These prior distributions are chosen as follows:

- $\mu \sim N(0, \psi_\mu)$
- $\boldsymbol{\alpha} \sim N(\mathbf{0}, \boldsymbol{I}_m)$
- $\boldsymbol{\rho}$, $\rho_l \sim U[0, 1], l = 1, \dots d$

These parameters are estimated using a MCMC [29] approach, namely Metropolis-Hastings, more specifically a Metropolis within Gibbs; the pseudo code can be found in Algorithm 8 and will be discussed in the next section. For further details, see Linkletter [47].

Latin Hypercube Design

In 1979, McKay et al. [52] introduced the LHD. The design points generated from a LHD guarantee that when projected in any dimension, they achieve complete stratification. Furthermore, the used implementation fulfills the space-filling criterion and ensures minimum distance between the design points [41]. In Figure 6.2, a two-dimensional example for 10 design points is shown.

Advantages of LSSP Model

The LSSP model is a state-of-the-art model for link prediction in FVN. One of the major improvements over the previous models (with exception of the logistic regression model) is the constant

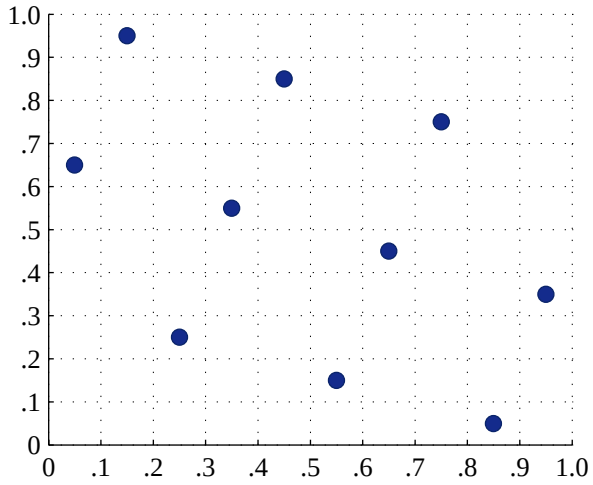


Figure 6.2: Two-dimensional space-filling Latin Hypercube Design.

number of parameters. In the LSSP model, the positions of actors are not directly estimated, but based on their respective feature vectors which avoids overfitting. This implies that this model can be used to make predictions for actors *like* this, but not for specific actors. In other words, actors are reduced to their feature vectors; therefore, actors with the same feature vectors are treated equally. One further advantage is that the position in social space of a new actor can be determined. This enables us to predict links for new actors. This is the only state-of-the-art model which has this capability. Currently, the z function which projects actors into the social space is one-dimensional. However, as mentioned in Linkletter [47] it is possible to extend this function to a higher dimensional cases.

6.3.2 Application of LSSP Model

The AddHealth dataset is a publicly available dataset[1] based on surveys of high school students. Each student was asked to name up to five friends. We considered a friendship link between two students if either of them named the other as his friend. Our particular subset contains 205 students and 203 friendship relationships, where 57 students did not report any friends. The dataset contains one-dimensional covariates, the grade of a student ranging from 7 to 12.

[1] http://www.cpc.unc.edu/projects/addhealth/data

Table 6.1: **AddHealth0:** parameter values for $\mu, \boldsymbol{\alpha}$ and $\boldsymbol{\rho}$.

μ	0
$\boldsymbol{\alpha}$	[1.61,1.86,1.64,1.38,1.18,1.09,0.72,-0.13,-0.28,-1.12]
$\boldsymbol{\rho}$	0.16

We applied the LSSP model to this dataset. Since a Bayesian approach was chosen, not a single value for the parameters was returned, but a posterior distribution. In order to estimate the z function and the link probabilities, we averaged over samples drawn from this distribution. The posterior mean z function (LSSP scores) can be found in Figure 6.3. The non-monotone graph reflects the fact that grade 10 and 12 students tend to be more likely to be friends with each other than grade 10 and 11 students. Sorting the posterior mean probabilities based on the LSSP scores of the respective actors results in the heat map of Figure 6.4. In this heat map, high probabilities are in red colors and low probabilities in blue colors. The sorting is based on the LSSP scores of the actors. Red and yellow areas reflect a high *clustering* of actors, *i.e.*, a high probability that people are connected. Light blue areas reflect a lower clustering of students. This information can be used to identify clusters in the data.

In the following, we sometimes need a single parameter value. In this case, we use the posterior mean parameter estimates (instead of the posterior distribution) of the AddHealth dataset. Also, sometimes we set μ to 0 in order to increase the number of friends to a more realistic number (in the AddHealth dataset on average each student had only one friend). We refer to this new setting as **AddHealth0**. Its parameter values can be found in Table 6.1.

6.3.3 *simLSSP*

In the following *simLSSP*, an algorithm for simulating FVNs based on the LSSP model, is introduced. Based on a given FVN the goal is to simulate FVNs which have similar statistical properties as a given input FVN $\mathcal{G}$. The first step is to estimate the parameters of the LSSP model for $\mathcal{G}$. Since a Bayesian approach was used, the result of this estimation consists not of single values for μ, $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$, but of samples from the posterior distribution. For example, $\boldsymbol{\alpha} = (\alpha^{(1)}, \ldots, \alpha^{(nMCMC)})$, where $nMCMC$ is the number of MCMC iterations, analogous for $\boldsymbol{\rho}$ and μ. For more details on the parameter estimation see Linkletter [47] and Section 6.4.4.

The pseudo code of *simLSSP* can be found in Algorithm 7. The algorithm assumes the following

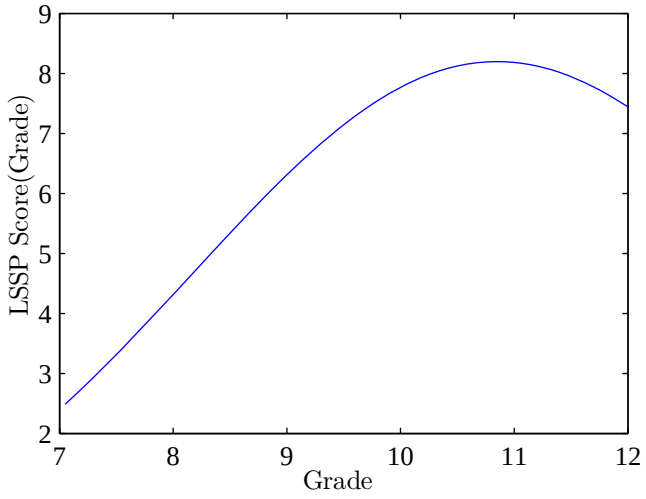


Figure 6.3: **AddHealth:** Grades and their respective LSSP scores.

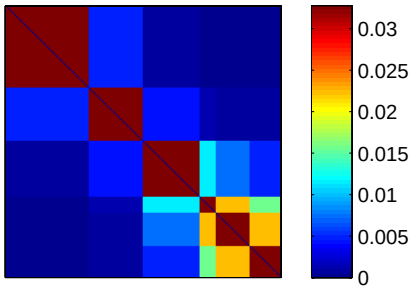


Figure 6.4: **AddHealth:** Sorted posterior mean probabilities.

input parameters: the new covariates $\boldsymbol{x_{new}}$, the distribution of the model parameters $\boldsymbol{\alpha}$, $\boldsymbol{\rho}$, μ, and the MCMC parameters $nMCMC$, $burnIn$ and $nGrab$. Depending on the desired size and properties of the simulated FVN there are several options for choosing $\boldsymbol{x_{new}}$. The first option is to use the covariates of $\mathcal{G}$. Another option is to sample from a distribution using the fitted covariates. As a third option, $\boldsymbol{x_{new}}$ can also be specified by the user.

Algorithm 7 ***simLSSP*****:** Simulation of FVNs using LSSP model

1: INPUT: $\boldsymbol{x_{new}}$, $\boldsymbol{\alpha}$, $\boldsymbol{\rho}$, $\boldsymbol{\mu}$, $nMCMC$, $burnIn$, $nGrab$
2: OUTPUT: simulated FVN $newFVN$
3: $n \leftarrow$ size($\boldsymbol{x}_{new}$)
4: $step = \lceil (nMCMC - (burnIn + 1))/nGrab \rceil$
5: // Go through MCMC samples
6: **for** ($t = burnIn + 1; t \leq nMCMC; t = t + step$) **do**
7: **for** (i = 1 to n) **do**
8: **for** ($j = i + 1$ to n) **do**
9: $z(\boldsymbol{x_{i,new}})^{(t)} \leftarrow \sum_r \alpha_r^{(t)} \prod_{l=1}^{d} (\rho_l^{(t)})^{(x_{il} - w_{rl})^2}$
10: $z(\boldsymbol{x_{j,new}})^{(t)} \leftarrow \sum_r \alpha_r^{(t)} \prod_{l=1}^{d} (\rho_l^{(t)})^{(x_{jl} - w_{rl})^2}$
11: $\pi_{i,j}^{(t)} \leftarrow ilogit(\mu^{(t)} - |z(\boldsymbol{x_{i,new}})^{(t)} - z(\boldsymbol{x_{j,new}})^{(t)}|)$
12: **end for**
13: **end for**
14: **end for**
15: // Simulate edges
16: **for** i = 1 to n **do**
17: **for** $j = i + 1$ to n **do**
18: $\pi_{i,j} \leftarrow mean(\pi_{i,j}^{(t)})$
19: **if** ($\pi_{i,j} \leq random(1)$) **then**
20: $newNetwork$.addEdge(i, j)
21: $newNetwork$.addEdge(j, i)
22: **end if**
23: **end for**
24: **end for**
25: **return** $newFVN$ = ($newNetwork, \boldsymbol{x}_{new}$)

The parameter $burnIn$ reflects the number of samples discarded and the parameter $nGrab$ reflects the number of samples taken into account. Using $\boldsymbol{\alpha}$ again as an example, the first considered sample is $\boldsymbol{\alpha}^{(burnIn+1)}$, the next sample is $\boldsymbol{\alpha}^{(burnIn+1+step)}$, where $step = \lceil (nMCMC - (burnIn + 1))/nGrab \rceil$. This results in $nGrab$ samples over which the average is calculated, see line 18 of Algorithm 7.

The algorithm starts with setting n to the number of feature vectors (which corresponds to the number of actors in the network) and specifying $step$. Next, going over all samples and all combinations of nodes, the LSSP scores for these nodes are calculated. Based on these LSSP scores,

we are able to estimate the link probabilities. In the second set of "for loops", the link probabilities are estimated by averaging over all samples. The edges are simulated based on these probabilities. The resulting network structure in combination with the given feature vectors provides the simulated FVN.

Observations of *simLSSP*

The algorithm *simLSSP* is able to simulate FVNs of various sizes and arbitrary covariate distributions. We have now a closer look at the average degree of actors in FVNs simulated by *simLSSP*, see Figure 6.5. We use the posterior means of $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$ from the **AddHealth** dataset and set μ to 0. Furthermore, the same number of students from each grade is used. Going back to the high school example from the introduction, if we apply *simLSSP* to a high school of 100 students, then the average number of friends is less than 25. Increasing the number of students to 2,000 this results, on average, in over 400 friends which is not very realistic. Next, we theoretically analyze the average degree of FVNs simulated by *simLSSP*.

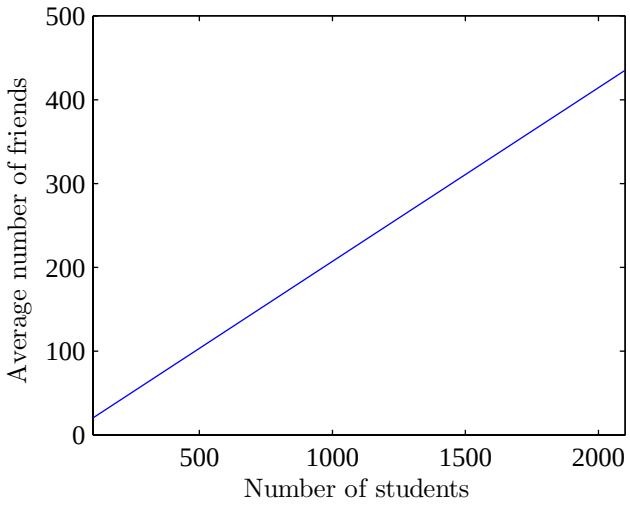


Figure 6.5: **AddHealth:** Simulation of FVNs of various sizes.
The values on the x-axis reflect the number of students. On the y-axis, the average number of friends per student in the by *simLSSP* simulated FVN can be found.

Expected Degree - Bernoulli Graph Model

Given a set of nodes, the Bernoulli graph model [16] is a statistical model which assumes a constant link probability for having an edge between any two nodes. It can be seen as an extreme case of the LSSP model in which the LSSP scores are the same for every possible covariate, see also Figure 6.6. Constant LSSP scores (independent of the covariate information) are a result of setting the $\boldsymbol{\alpha}$'s to $\vec{0}$ and/or the $\boldsymbol{\rho}$'s to $\vec{1}$. The expected degree in the Bernoulli graph model is $n\pi$, where n is the number of nodes in the graph and π the link probability. In the LSSP model $\pi = ilogit(\eta) = ilogit(\mu)$ if $z(\boldsymbol{x_i}) = z(\boldsymbol{x_j}) \forall i, j$. Furthermore, μ is constant and not dependent on n. Therefore, the expected degree in the LSSP model with constant LSSP scores is: $n \cdot ilogit(\mu)$.

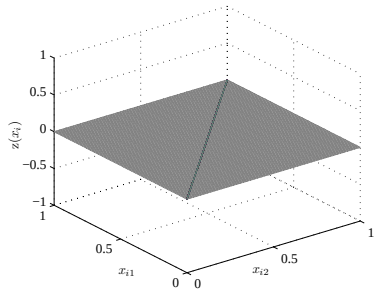


Figure 6.6: Constant LSSP scores.
Parameter values: $\boldsymbol{\alpha} = \vec{0}$ and/or $\boldsymbol{\rho} = \vec{1}$.

Expected Degree - LSSP Model

Recall that $\pi_{i,j}$ denotes the probability that actor i and j are connected in the LSSP model. Since the model assumes the links to be independent, the expected degree of node i in a network of size n is $\sum_{k=1}^{n} \pi_{i,k}$. Since $\pi_{i,k}$ is bound by $ilogit(\mu)$, the upper bound for the expected degree is

$$n \cdot ilogit(\mu).$$

However, the absolute distance in the social space $|z(\boldsymbol{x_i}) - z(\boldsymbol{x_j})|$ is also independent of n. Therefore, for a given pair of actors their distance in social space is not affected by the network size. We call this phenomenon *constant distance in social space*.

Since μ is constant, the network size has no impact on the $\pi_{i,j}$'s. Assuming that the covariates are generated from the same distribution (and this distribution is independent of the sample size), we can infer that $\sum_{k=1}^{n} \pi_{i,k}$ is linear in n as well. We call this phenomenon *constant link probability*.

In summary, we identified the following two undesired phenomena of *simLSSP*:

- **Constant Link Probability**
  The expected degree of an actor grows linearly with the number of nodes in the network if the covariates are simulated from the same distribution. We refer again to the example in Figure 6.5.

- **Constant Distance in Social Space**
  The social space is independent of the size of the network. If the number of potential friends is very low, people do not have a large selection in whom they choose as friends. However, if the population size is larger, then people have a larger chance to meet individuals who are similar to them. Given that people have a limit on how many friends they can have, the larger the "choice" of friends, the more likely people can become friends with similar individuals [46].

The n^*LSSP model, introduced in the next section, addresses these problems. However, it is flexible enough to model the constant link probability and constant distance in social space as one of its special cases.

6.4 n^*LSSP Model

In this section, the n^*LSSP model, an extension of the LSSP model, is introduced. The n^*LSSP model addresses the constant link probability and the constant distance in social space by introducing two dampening functions. The new model is shown and its parameter estimation explained. Last, the algorithm *simn*LSSP*, which can simulate FVNs of arbitrary size, is introduced.

Base Network

In the following, we define a *base network*. A base network has two purposes: first, it helps to make the parameters of the n^*LSSP model identifiable. Second, it ties the n^*LSSP model back to the LSSP model. Formally, it is defined as:

Definition 6.1 (Base Network) *A **base network**, denoted by G_{base}, is a network of size $n_{base} = size(G_{base})$ such that the n*LSSP and LSSP model are exactly the same for G_{base}.*

Typically n_{base} is set to the size of the smallest conceivable network. We could have chosen $n_{base} = 1$. However, the introduced definition will make more sense, once the n^*LSSP model is introduced. By choosing this definition, a very intuitive interpretation of the n^*LSSP model is provided as we will see in Section 6.4.3.

6.4.1 Flexible Link Probability

Recall that in the previous section, we showed that the link probability is independent of the network size which might cause undesired effects, when simulating networks of larger sizes. The *flexible link probability* is introduced as a cure for the constant link probability. The $\pi_{i,j}$ reflects the probability that actor i and j are connected with each other. It is independent of the network size, since μ, the function z and the covariates $\boldsymbol{x_i}$, $\boldsymbol{x_j}$ are independent of the network size. In the following, we like to make $\pi_{i,j}$ dependent on the network size, which we denote with n^*. We propose to include a **dampening function**, $d(n^*)$, to $\pi_{i,j}$. The function d can be chosen in many ways, we elect to use $d(n^*) = \left(\frac{n_{base}}{n^*}\right)^l$. We denote the link probability now with $\pi_{i,j}^{n^*}$ and define it as:

$$\pi_{i,j}^{n^*} = d(n^*)\pi_{i,j} = \left(\frac{n_{base}}{n^*}\right)^l \pi_{i,j}.$$

The variable l is a parameter and can either be estimated or specified by the user. We restrict the parameter l to be in the interval [0,1]. The reason for this is as follows: we want the term $\left(\frac{n_{base}}{n^*}\right)^l$ to be between 0 and 1. If l was smaller than 0, then the link probability would increase with the network size. In other words, actor i and j would be more likely to be connected in a larger network than in a smaller network. However, this is different from our goal, namely the more people in the network, the less likely people are connected. In case l was larger than 1, the average degree of a person would be decreasing with the network size. Meaning, if in a network of size **20** on average a person has **four** friends, in a network of size **40**, this person might have only **three**. Whereas this might be desirable in some cases, it is not ours.

The term $\pi_{i,j}^{n^*}$ represents the probability for i and j being connected in a network of size n^*. One of the requirements is therefore that it is between 0 and 1 which we show in the following:

- The first term, $\left(\frac{n_{base}}{n^*}\right)^l$, is between 0 and 1 for $n^* \geq n_{base}$ and $l \geq 0$.
- The second term, $\pi_{i,j}$, is between 0 and 1 due to the logistic transformation of $\eta_{i,j}$.

- The product of two terms between 0 and 1 is between 0 and 1 as well.

This results in:

$$\begin{aligned} Pr(Y_{i,j} = 1) &= \pi_{i,j}^{n^*} \\ Pr(Y_{i,j} = 0) &= 1 - \pi_{i,j}^{n^*}. \end{aligned}$$

Parameter-Sensitivity of Flexible Link Probability

In order to provide more intuition about the meaning of the new parameter l, in Figure 6.7(a), we depicted the graphs of the dampening function $d(n^*) = \left(\frac{n_{base}}{n^*}\right)^l$ for several values of l (0, 0.25, 0.5 0.75 and 1). The parameter n_{base} is set to 50 and n^* is between 50 and 550. The blue graph represents $l = 0$. In this case, the function is always 1. This implies that dampening function has no impact and the link probability is still constant, *i.e.*, the model is equivalent to the LSSP model. On the other hand in case $l = 1$ (purple graph), the function decreases with n^*; for $n^* = 50$ it equals to 1, for $n^* = 100$ it equals to $\frac{1}{2}$ and so on. For l values between 0 and 1, the results are between the constant link probability and the one for $l = 1$. Therefore, the term $\left(\frac{n_{base}}{n^*}\right)^l$ dampens the link probability and enables us to control the expected degree in a simulated network as shown in Section 6.4.5.

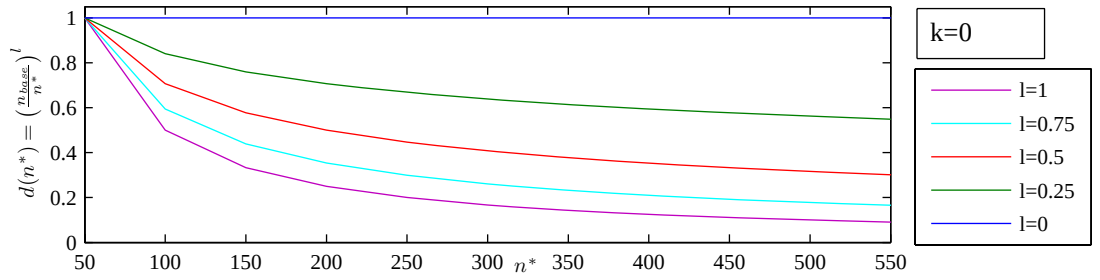
6.4.2 Flexible Distance in Social Space

In a remote, very small high school of let's say 20 students, almost everybody knows everybody - independent of their feature vectors, such as age and gender. However, the larger the choice for forming friendships, the more likely students are to be friends with *similar* students. In a high school of 500 people, it is not very likely that all students know each other.

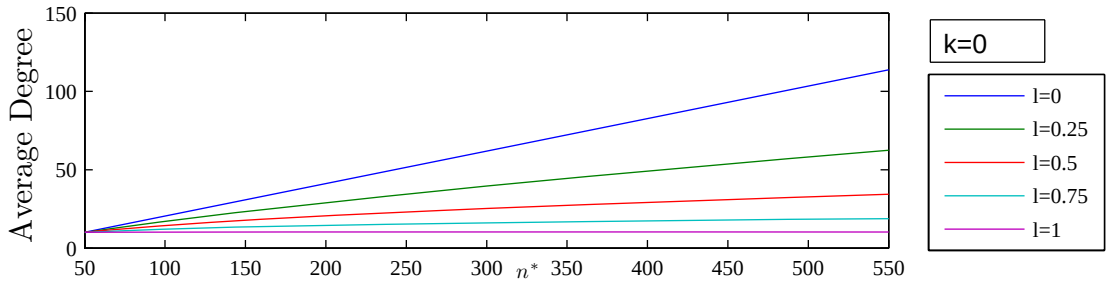
Therefore, we propose to adjust the LSSP model by adding a dampening function to the distances in the social space, called **distance dampening function** and denoted with dd. We chose $dd(n^*) = \left(\frac{n^*}{n_{base}}\right)^k$ as distance dampening function. Therefore,

$$\eta_{i,j}^{n^*} = \mu - dd(n^*)|z(\boldsymbol{x_i}) - z(\boldsymbol{x_j})| = \mu - \left(\frac{n^*}{n_{base}}\right)^k |z(\boldsymbol{x_i}) - z(\boldsymbol{x_j})|.$$

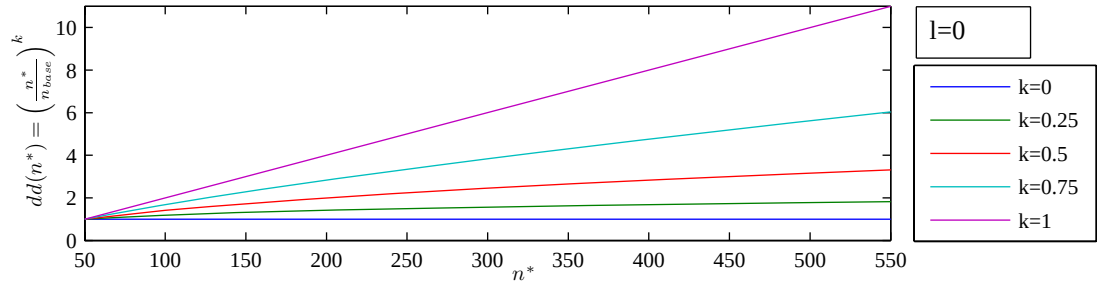
Again n^* denotes the network size and k is a parameter. The goal is to increase the distance in social space when increasing the number of people. For $k < 0$, the distance in social space would decrease with increasing number of people. Therefore, we require $k \geq 0$. On the other hand, if $k > 1$, then - depending on the covariates and the values of $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$ - the average degree



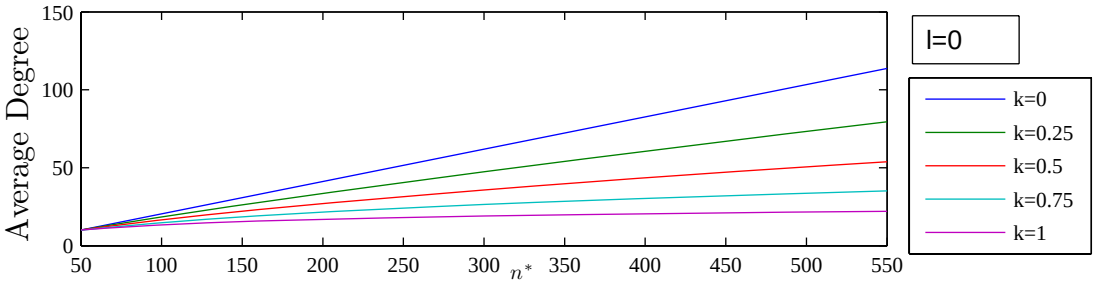
(a) Dampening function $d(n^*)$.



(b) Average degree of nodes in a FVN simulated by *simn*LSSP*.



(c) Distance dampening function $dd(n^*)$.



(d) Average degree of nodes in a FVN simulated by *simn*LSSP*.

Figure 6.7: ***n**LSSP model**: $n_{base} = 50$.

might indeed decrease with increasing network size which is not desired in our context. Intuitively, the more people in the social network, the larger their distance in social space, since the capacity of making friends is limited. If people have a larger choice in terms of friendships, they tend to befriend more similar people. We model this by multiplying the absolute distance, $|z(\boldsymbol{x_i}) - z(\boldsymbol{x_j})|$, with the term $\left(\frac{n^*}{n_{base}}\right)^k$ which grows with n^*. Overall, the distance dampening function governs the distance in social space, whereas the dampening function governs the overall probability.

Parameter-Sensitivity of Flexible Distance in Social Space

Again, we provide some more intuition in form of a figure. In Figure 6.7(c), the graphs of the function $\left(\frac{n^*}{n_{base}}\right)^k$ are depicted for several values of k (0, 0.25, 0.5 0.75 and 1). The parameter n_{base} is set to 50 and n^* is between 50 and 550. The blue graph represents $k = 0$. In this case the function is always 1. This implies that the social space is still constant, since the distance dampening function has no impact and the model is still equivalent to the LSSP model. On the other hand, in case $k = 1$ (purple graph), the term increases linearly with n^*; for $n^* = 50$ it equals to 1, for $n^* = 100$ it equals to 2 and so on. For k values between 0 and 1, the results are between the constant distance in social space and the one for $k = 1$. Therefore, the introduced distance dampening function helps us to control the distances in social space based on the network size.

6.4.3 Model

We combine the two previously introduced concepts (flexible link probability and flexible distance in social space) into a single model and call this new model n^*LSSP model. The n^*LSSP model takes into consideration not only the overall decrease in link probability with increasing network size, but also the increase in distances in social space. Using a slightly different notation (the link probability is now denoted with $\pi_{i,j}^{n^*}$), the overall probability of a socio matrix $\boldsymbol{y}$ is again:

$$Pr(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{x}) = \prod_{i>j} Pr(Y_{i,j} = y_{i,j}|\boldsymbol{\theta}, \boldsymbol{x_i}, \boldsymbol{x_j}) =$$

$$\prod_{i>j} (\pi_{i,j}^{n^*})^{y_{i,j}} (1 - \pi_{i,j}^{n^*})^{1-y_{i,j}}$$

The link probability, $\pi_{i,j}^{n^*}$, is calculated as follows:

$$\pi_{i,j}^{n^*} = \left(\frac{n_{base}}{n^*}\right)^l p_{i,j}^{n^*},$$

where

$$p_{i,j}^{n^*} = ilogit(\eta_{i,j}^{n^*})$$

and

$$\eta_{i,j}^{n^*} = \mu - \left(\frac{n^*}{n_{base}}\right)^k |z(\boldsymbol{x_i}) - z(\boldsymbol{x_j})|.$$

Therefore, the model combines the idea of a flexible link probability with the one of the flexible distance in social space. Next, the parameter estimation of n^*LSSP is discussed.

6.4.4 Parameter Estimation

In addition to the existing parameters of the LSSP model, namely $\boldsymbol{\alpha}$, $\boldsymbol{\rho}$ and μ, the parameters l and k have to be estimated. Therefore, in total, $11d+3$ parameters need to be estimated. These parameters are estimated using MCMC. In order to be able to estimate l and k at least two input networks are necessary as argued in the following.

Number of Input FVNs Required for the n^*LSSP Model

Recall the definition of the latent function z:

$$z(\boldsymbol{x_i}) = \sum_{r=1}^{m} \alpha_r \prod_{l=1}^{d} \rho_l^{(w_{rl}-x_{il})^2}.$$

The absolute distance in social space is calculated as

$$|z(\boldsymbol{x_i}) - z(\boldsymbol{x_j})| = \left| \sum_{r=1}^{m} \alpha_r \left(\prod_{l=1}^{d} \rho_l^{(w_{rl}-x_{il})^2} - \prod_{l=1}^{d} \rho_l^{(w_{rl}-x_{jl})^2} \right) \right|.$$

This distance multiplied with a constant $c > 0$ results in

$$c|z(\boldsymbol{x_i}) - z(\boldsymbol{x_j})| = \left| \sum_{r=1}^{m} c\alpha_r \left(\prod_{l=1}^{d} \rho_l^{(w_{rl}-x_{il})^2} - \prod_{l=1}^{d} \rho_l^{(w_{rl}-x_{jl})^2} \right) \right|.$$

It is clear that c and α_r are not jointly identifiable. Considering $c = \left(\frac{n^*}{n_{base}}\right)^k$ makes k unidentifiable. A similar argument can be made for l. Therefore, it is necessary to have at least two input FVNs. The sufficient number of input FVNs is determined experimentally later on in this chapter. In the following, we assume that the number of input FVNs is larger than one.

Prior distributions of l and k

Based on the constraints on l and k we set the prior distribution to a uniform distribution in the interval [0,1].

Relationship between LSSP and n^*LSSP Model

The connection between the LSSP and n^*LSSP model is with the base network. The n^*LSSP model has the following form for $n^* = n_{base}$:

$$\eta_{i,j}^{n^*} = \mu - \left(\frac{n^*}{n_{base}}\right)^k |z(\boldsymbol{x_i}) - z(\boldsymbol{z_j})|$$

which results for $n^* = n_{base}$ in:

$$\eta_{i,j}^{n^*} = \mu - \left(\frac{n_{base}}{n_{base}}\right)^k |z(\boldsymbol{x_i}) - z(\boldsymbol{x_j})| = \mu - |z(\boldsymbol{x_i}) - z(\boldsymbol{x_j})| = \eta_{i,j}.$$

The link probability $\pi_{i,j}^{n^*}$ is:

$$\pi_{i,j}^{n^*} = \left(\frac{n_{base}}{n^*}\right)^l p_{i,j}^{n^*} =$$

$$\left(\frac{n_{base}}{n_{base}}\right)^l p_{i,j}^{n^*} = \left(\frac{n_{base}}{n_{base}}\right)^l ilogit(\eta_{i,j}) = ilogit(\eta_{i,j}) = \pi_{i,j}.$$

Thus,

$$\pi_{i,j}^{n^*} = \pi_{i,j}.$$

Therefore, the n^*LSSP model is indeed the same as the LSSP model for $n^* = n_{base}$. In other words, for the base network, the dampening functions (link dampening and dampening of distance in social space) are inactive.

Algorithm

The algorithm for the parameter estimation can be found in Algorithm 8. We use Metropolis-within-Gibbs Sampling [29] for estimating the parameters. As input, it requires $numberNetworks > 1$ input FVNs; the number of iterations, $nMCMC$; and the size of the base network, n_{base}. The parameter n_{base} is required to be smaller or equal to the size of the smallest input network. First, the parameters and loglikelihood are initialized (line 3). Then in each iteration, the parameters are sequentially updated (based on their proposal distribution) and - depending on the loglikelihood of the model and the proposal distribution - accepted or rejected.

Algorithm 8 ***n****LSSP** Parameter Estimation (Metropolis-within-Gibbs Sampling)

INPUT: $FVN^* = \{FVN_1, \ldots FVN_{nrNetworks}\}$, $nMCMC$, n_{base}
OUTPUT: $\mathbf{\Theta}$
Initialize $\mathbf{\Theta}^{(0)}$, $L^{(0)}$
for I = 1 to $nMCMC$ **do**
  $L^{(I)} = L^{(I-1)}$
  $\mathbf{\Theta}^{(I)} = \mathbf{\Theta}^{(I-1)}$
  for ($\theta \in \{\mu, \alpha, \rho, l, k\}$) **do**
    $\theta_{new} \leftarrow$ draw from proposal of θ given $\theta^{(I)}$
    for (J = 1 to $nrNetworks$) **do**
      L(J) = likelihood($FVN_J | \mathbf{\Theta}^{(I)} \backslash \{\theta^{(I)}\} \cup \{\theta_{new}\}, covariates(FNV_J)$)
    end for
    L_{new} = $\sum(L(J))$
    if ($(L_{new} - L^{(I)} + L(\theta_{new}) - L(\theta_I)) < log(rand(1))$) **then**
      // Accept proposal
      $\theta^{(I+1)} = \theta_{new}$
      $L^{(I)} = L_{new}$
      $\mathbf{\Theta}^{(I)} = \mathbf{\Theta}^{(I)} \backslash \{\theta^{(I)}\} \cup \{\theta_{new}\}$
    else
      // Reject proposal
      $\theta^{(I+1)} = \theta^{(I)}$
    end if
  end for
end for
return $\mathbf{\Theta}^{(1 \ldots nMCMC)}$

We explain this procedure now in more detail using $\theta = \mu$. First, a new value for μ is drawn from its proposal distribution given the value of μ during the previous iteration. For each network its loglikelihood under consideration of the current parameter setting - replacing the value for μ by the new value - is calculated. Afterwards, these likelihoods are summed up, resulting in L_{new}. The new value for μ is accepted, respectively rejected, based on the difference between likelihoods for the networks and the loglikelihoods with which the old and new value of μ are drawn from the proposal distribution given the old value of μ.

6.4.5 *simn*LSSP*

Next, we introduce *simn*LSSP*, an algorithm for simulating FVNs of arbitrary sizes. This algorithm is based on the *n**LSSP model. The pseudo code can be found in Algorithm 9. In addition to the parameters of the LSSP model, the parameter n_{base}, l and k are required. The first step is to use the *n**LSSP model to estimate the model parameters based on several FVNs. The posterior distributions of the parameters are used as input for the *simn*LSSP* model.

Algorithm 9 ***simn***LSSP*:** Simulation of FVNs using n^*LSSP model

INPUT: $\boldsymbol{\alpha}$, $\boldsymbol{\rho}$, $\boldsymbol{\mu}$, l, k, n_{base} $\boldsymbol{x}_{new}$, $nMCMC$, $burnIn$, $nGrab$
OUTPUT: simulated FVN $newFVN$
$n^* \leftarrow$ size($\boldsymbol{x_{new}}$)
$step = \lceil (nMCMC - burnIn + 1)/nGrab \rceil$
// Go through MCMC samples
for ($t = burnIn + 1; t \leq nMCMC; t = t + step$) **do**
  for i = 1 to n^* **do**
    for $j = i + 1$ to n^* **do**
      // Update position in social space
      $z(\boldsymbol{x_{i,new}})^{(t)} \leftarrow \sum_r \alpha_r^{(t)} \prod_{l=1}^{d} \left(\rho_l^{(t)}\right)^{(x_{il} - w_{rl})^2}$
      $z(\boldsymbol{x_{j,new}})^{(t)} \leftarrow \sum_r \alpha_r^{(t)} \prod_{l=1}^{d} \left(\rho_l^{(t)}\right)^{(x_{jl} - w_{rl})^2}$
      // Update link probabilities
      $\eta_{i,j}^{n^*(t)} \leftarrow \mu^{(t)} - \left(\frac{n^*}{n_{base}}\right)^k |z(\boldsymbol{x_{i,new}})^{(t)} - z(\boldsymbol{x_{j,new}})^{(t)}|$
      $\pi_{i,j}^{n^*(t)} \leftarrow \left(\frac{n_{base}}{n^*}\right)^l ilogit\left(\eta_{i,j}^{n^*(t)}\right)$
    end for
  end for
end for
// Simulate edges for each combination of nodes
for i = 1 to n **do**
  for $j = i + 1$ to n **do**
    $\pi_{i,j}^{n^*} \leftarrow mean(\pi_{i,j}^{n^*(t)})$
    if ($\pi_{i,j}^{n^*} \leq random(1)$) **then**
      $newNetwork$.addEdge(i, j)
      $newNetwork$.addEdge(j, i)
    end if
  end for
end for
return $newFVN$ = ($newNetwork, \boldsymbol{x}_{new}$)

Next, we discuss the average degree in a FVN simulated by this new model. We simulated each 100 graphs of size 50 to 550 using one-dimensional equi-distant covariates. The rest of the parameters are the ones from **AddHealth0**. The results can be found in Figure 6.7(b). Keeping $k = 0$, for $l = 0$, the average degree is proportional to the network size; for $l = 1$ the average degree is forced to be constant. Therefore, the introduced dampening function helps to control the expected degree in a simulated FVN. In Figure 6.7(d), $l = 0$ and k is between 0 and 1. For $k = 0$ the degree is proportional to the FVN size. For $k = 1$ the degree stays almost constant.

6.5 Experiments

The goal of this section is to explore the *n**LSSP model and our methodology for simulating FVNs. Recall that the number of parameters in the *n**LSSP model is $11d + 3$, where d is the number of dimensions of the covariates. First, we provide some thorough analysis for the case of one-dimensional covariates. Afterwards, we show that the model and methodology also work for multi-dimensional cases.

6.5.1 One-Dimensional Covariates

In our preliminary analysis, we run experiments for many combinations of l and k. We observed that the closer l and k are to their boundaries (0 and 1), the better the results. We show in the following that even for the worst case performance ($l = k = 0.5$) the results are still pretty good.

Our experiments are based on synthetic datasets using *simn*LSSP*. It is critical to note that the parameters used for simulating the dataset, are not used during the estimation. In order to simulate the datasets under consideration, we used the following setup and call the dataset **1dim**. A summary of the parameter values can be found in Table 6.2.

n_{base}	60
$\mu, \boldsymbol{\alpha}, \boldsymbol{\rho}$	see Table 6.1
l, k	$l = k = 0.5$
Network sizes	60, 80, 100, 120
Covariates	1-dim (LHD plus noise)

Table 6.2: **1dim:** Parameters for dataset generation.

The size of base network, n_{base}, was set to 60. Instead of using samples of a posterior distribution, we used a single value for each parameter. The parameters for $\mu, \boldsymbol{\alpha}, \boldsymbol{\rho}$ were chosen as in Table 6.1; the parameters l and k were set to 0.5. The generated networks have size 60, 80, 100 and 120. Moving on the the covariates, the covariates were generated using a space-filling LHD. Afterwards, the minimum distance d_{min} between any two of the generated data points was calculated; we added uniform noise in the range of $[-d_{min}/2; d_{min}/2]$ to the sampled datapoints.

The general goal is to examine how well the parameters can be re-estimated for the set of datasets **1dim**. In particular, we are interested, how many FVNs are necessary and of what size they need to be. In case the dataset consisted of two FVNs (**1dim-2**), we used one FVN of size 60 and one of size 80. For the datasets containing three FVNs (**1dim-3**), we added one FVN of size 100 and for four

FVNs (**1dim-4**), we add one of size 120. As shown before, for a single network, the parameters l and k are under-specified. In total, for each type (**1dim-2**, **1dim-3** and **1dim-4**), we generated each 50 datasets. This results in 150 datasets for which we run experiments. Next, some of the trace plots for these experiments are discussed.

Trace Plots

In order to give some intuition about the convergence of the MCMC, we focus on three randomly chosen datasets (one from **1dim-2**, one from **1dim-3** and one from **1dim-4**) and show their parameter estimations. In Figure 6.8, Figure 6.9 and Figure 6.10, the trace plots for μ, $\boldsymbol{\rho}$, $\boldsymbol{\alpha}$, l (blue curve) and k (green curve) for $m = 10$ are depicted. The number of MCMC iterations is 100,000. We observe that the number of FVNs in the dataset has an impact on the variance of the estimates. The more FVNs, the lower variance of the estimates. Another interesting observation is that for **1dim-2**, the trace plots for l and k seem to be almost uniformly distributed between 0 and 1. However, one characteristic of MCMC estimates is that even if something like this happens, the overall result can be still very good. In our case, we mainly care about the link probabilities and not the parameter estimates. We show next that the link probabilities indeed converge.

In Figure 6.11, the probabilities are depicted. In the first column, the results for 2 FVNs are depicted, in the second for 3 and in the third one for 4. The first row shows the result for $n^* = 60$, the second for $n^* = 80$, third $n^* = 100$ and forth $n^* = 120$. We can see that the probabilities converge and an increasing number of networks results in a lower variance of the probabilities.

Variance of Parameter Estimates

Given the 100,000 samples from the posterior distribution, for the following evaluation, we use a burn-in of 10,000 and use every 9th sample (which results 10,000 samples). For each dataset, we calculate the posterior mean of μ, l and k. In Figure 6.12, the histograms of these posterior means are shown. The red graph denotes the prior distribution; the red triangle the true parameter value. In Figure 6.13, the standard deviations for **1dim** are depicted. The variance of the parameters l and k clearly decreases with the number of FVNs. This is due to the fact that the number of FVNs can be considered as the number of observations from which these parameters are estimated. As shown before, they are not identifiable for a single FVNs. The reason, why the variance of μ decreases with the number of FVNs is very different.

Every observed link has an impact on the estimate of μ. For a dataset from **1dim-2**, the number

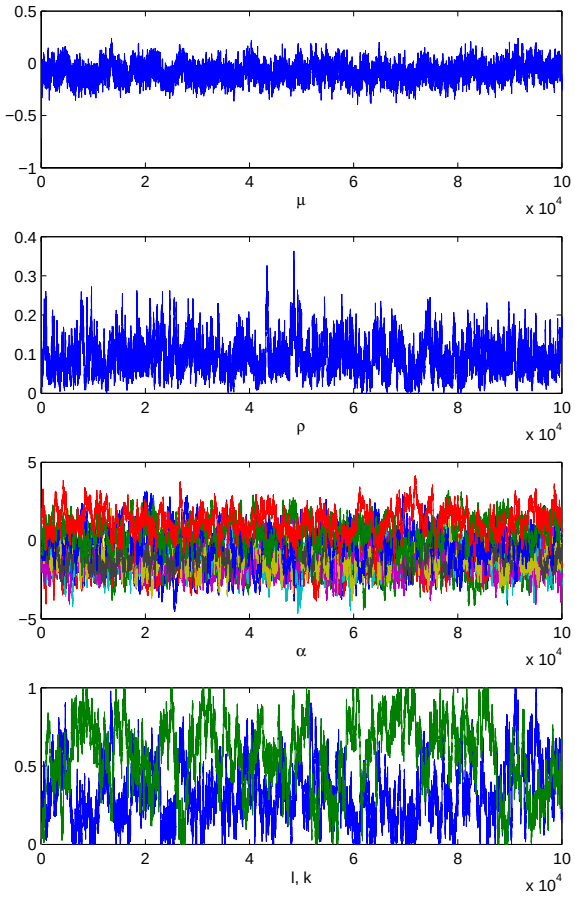


Figure 6.8: **1dim-2:** Trace plots of parameters. l (blue) and k (green).

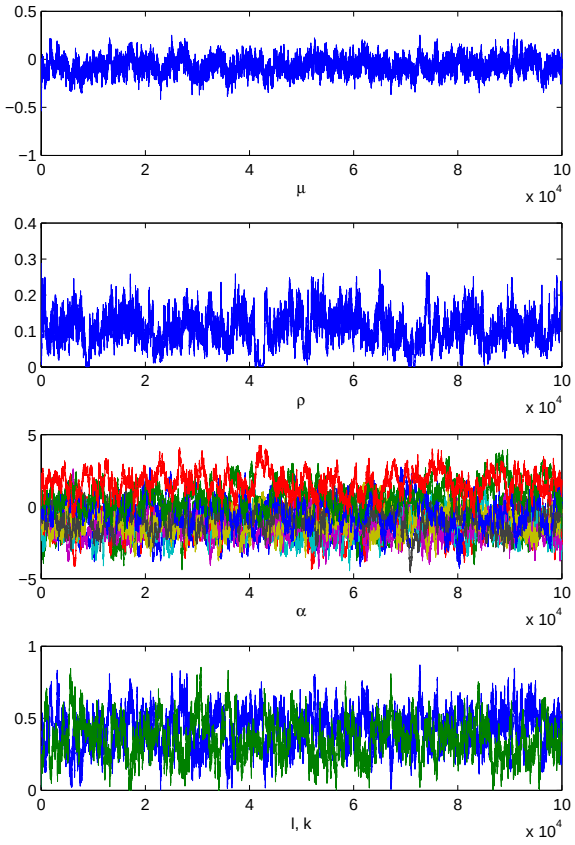


Figure 6.9: **1dim-3:** Trace plots of parameters. l (blue) and k (green).

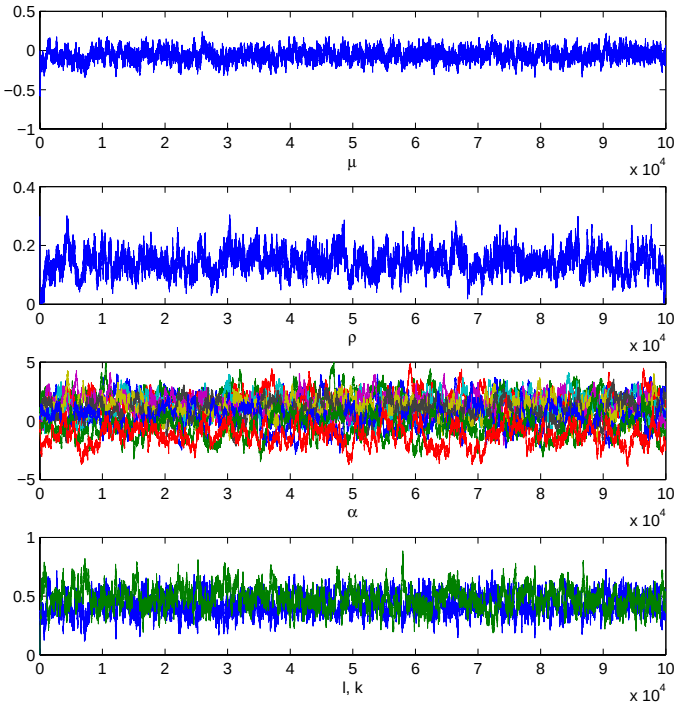


Figure 6.10: **1dim-4:** Trace plots of parameters.
l (blue) and k (green).

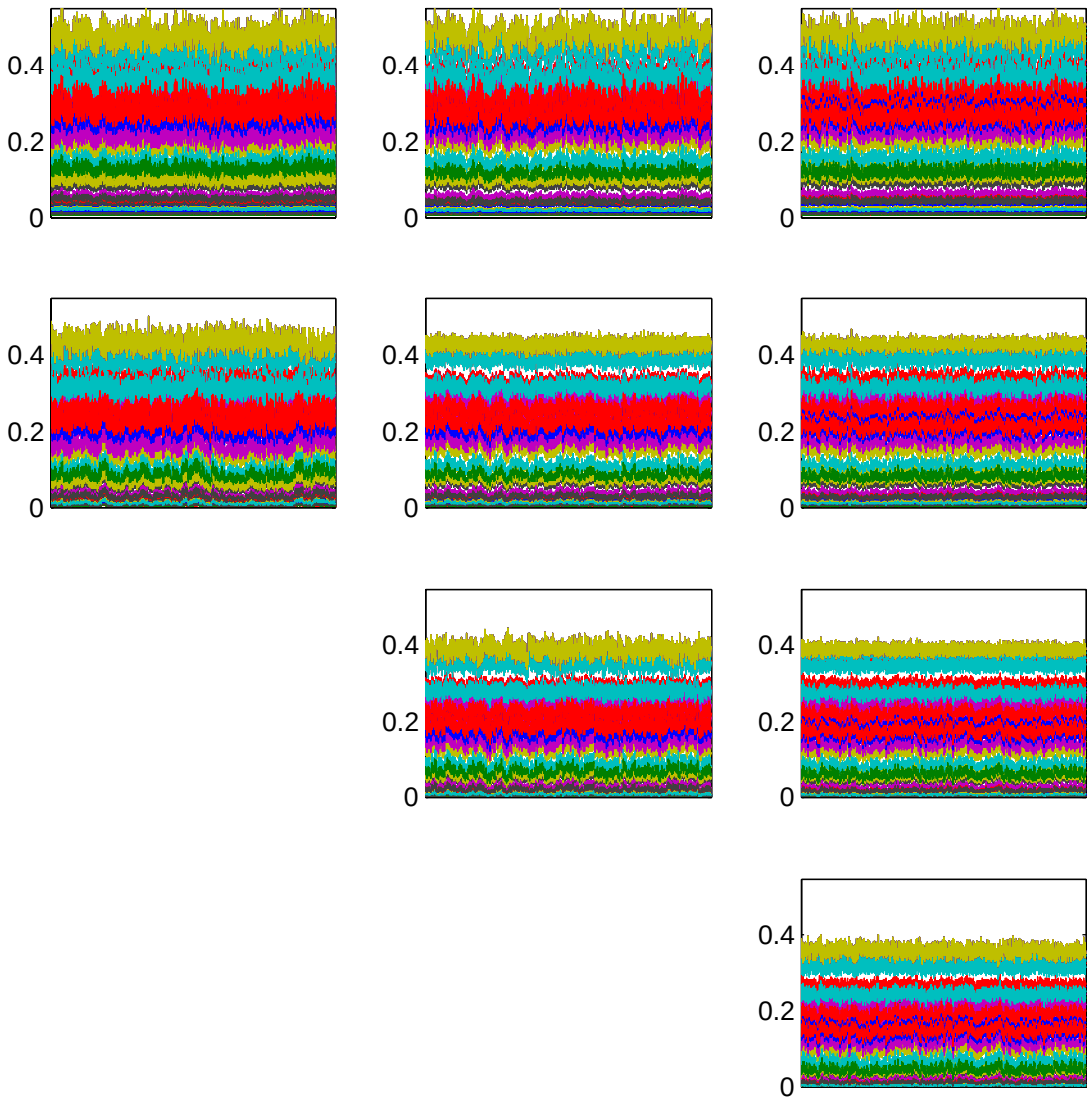


Figure 6.11: **1dim:** Trace plots of link probabilities.
Each column represents one dataset. The first column contains the results for a dataset from **1dim-2**; the second column the results for a dataset from **1dim-3** and the third the results for a dataset from **1dim-4**. The first row depicts the probabilities for the network of size 60, the second for the network of size 80, the third for size 100 and the forth for network of size 120.

of observations is $60 * 59/2 + 80 * 79/2 = 4930$, whereas it increases by $100 * 99/2 = 4950$ in **1dim-3**. Therefore, the decrease in variance is due to the increase in the number of observations.

Quality of Posterior Mean Link Probabilities

The evaluation of the link probabilities can be restricted to every combination of design points. The reason for that is that the $\boldsymbol{\alpha}$'s are design point specific. Since the link probabilities are symmetrical and there are $m = 10$ design points, this results in 55 link probabilities. The posterior mean link probabilities are compared with the true link probabilities based on the true parameter values.

We use two different types of diagrams: stem diagrams and boxplots. In the stem diagrams, the stems and circles represent the link probability calculated based on the true parameter values. The red triangle is the average over the posterior mean link probabilities over all datasets. In the boxplots, each box corresponds to the posterior mean link probability associated with two design points. The position of a box is determined based on the true link probability. Therefore, the closer the boxes to the black graph (identity function), the better the estimates.

In Figure 6.14, the stem diagrams for **1dim-2** can be found. We observe that the mean of the estimated link probabilities is extremely close to the true probabilities. In the left plot, we show the link probabilities for $n^* = n_{base} = 60$. In the right plot ($n^* = 80$) the effect of the dampening functions can be seen. In general, the probabilities are smaller. However, they are still very well estimated despite the fact that the trace plots of l and k were not close to the true parameter value, see Figure 6.8.

To provide more insight into the estimation of the link probabilities, we show their variability by providing boxplots. On the x-axis the true link probabilities are shown, on the y-axis the estimates. In Figure 6.15, the boxplots for **1dim-2** are shown, in Figure 6.16 the ones for **1dim-3** and in Figure 6.17 the ones for **1dim-4**. The estimates are very close to the true estimates and - as expected - the higher the number of FVNs in the dataset, the better the estimates. It is interesting to see that the estimates are better the closer they are to 0. They become worse, the closer they get to 0.5. The reason for that is the logistic link function which is more sensitive to probabilities around 0.5 than to probabilities close to 0 or 1.

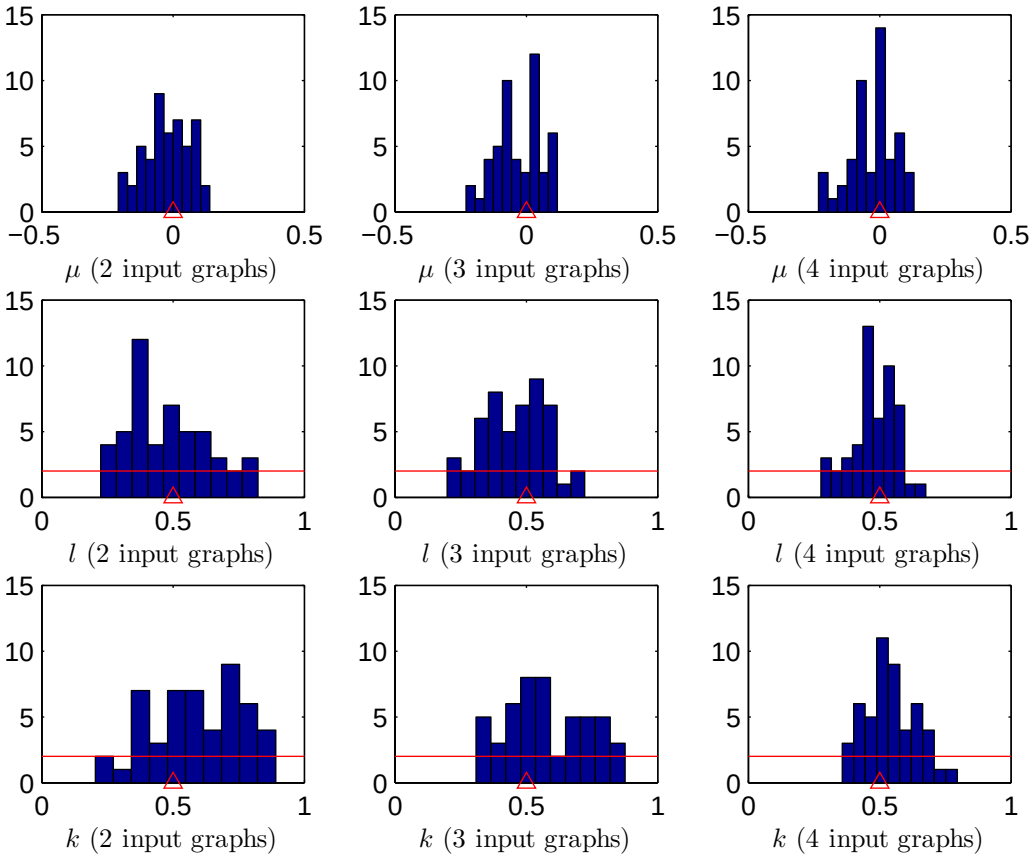


Figure 6.12: **1dim:** Histograms of posterior means of μ, l and k.
Red graph represents prior distributions. Red triangle true parameter value.

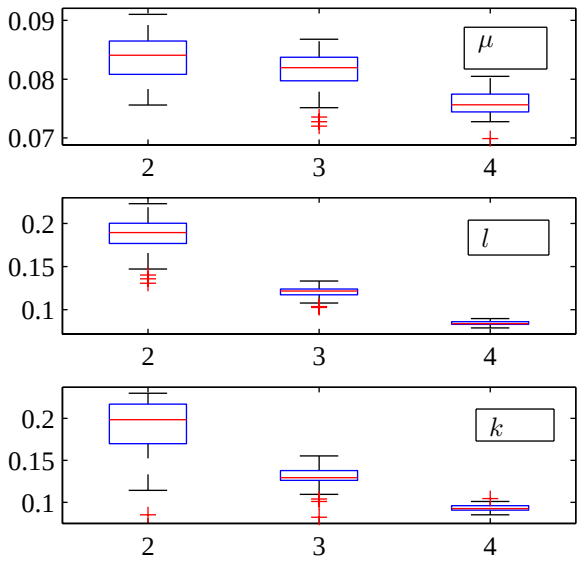


Figure 6.13: **1dim:** Boxplots of standard deviations for μ, l and k.
Number of FVNs in each dataset is shown on x-axis.

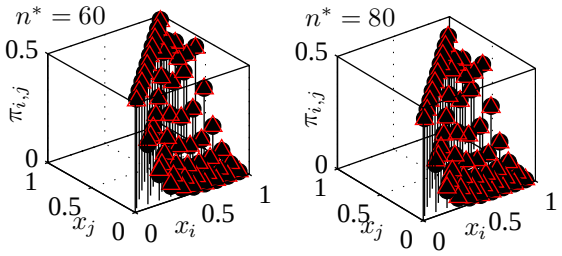


Figure 6.14: **1dim-2:** True link probabilities.
In black: true link probabilities. In red: mean of posterior mean link probabilities.

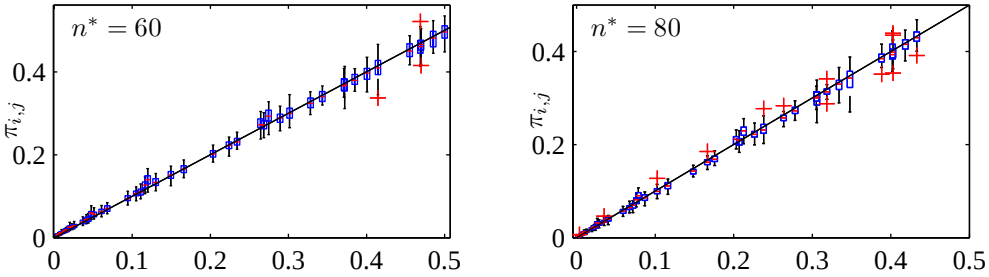


Figure 6.15: **1dim-2:** Boxplot of posterior mean probabilities.

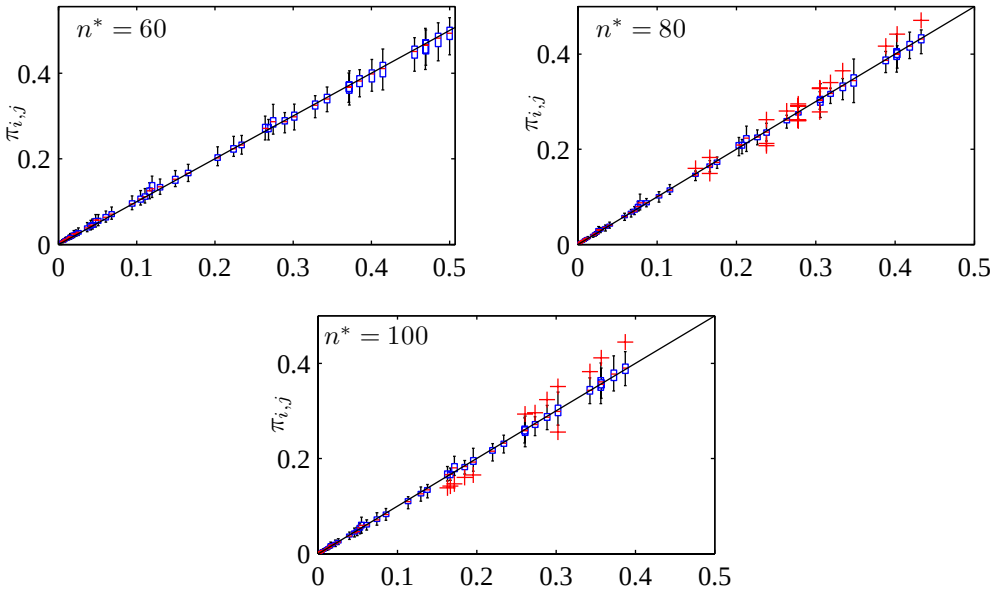


Figure 6.16: **1dim-3:** Boxplot of posterior mean probabilities.

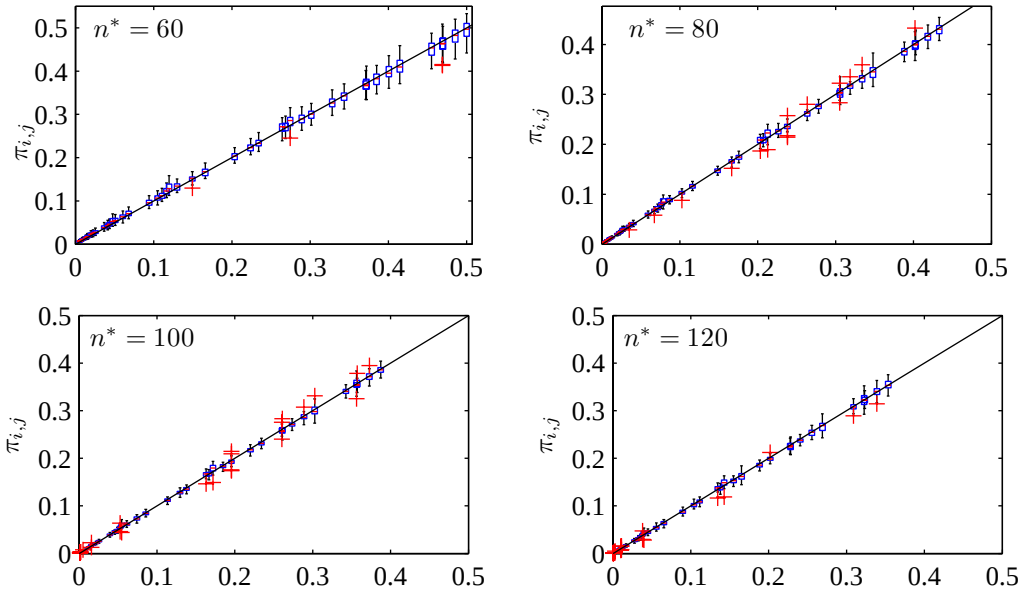


Figure 6.17: **1dim-4:** Boxplot of posterior mean probabilities.

6.5.2 Multi-Dimensional Covariates

In the following, we assume again that FVNs are given. We use two examples, one with 3-dimensional and one with 5-dimensional covariates. The FVNs are simulated using *simn***LSSP*.

The following dataset is called **3dim**. The critical parameters are again μ and the $\boldsymbol{\rho}$'s. However, in order to simulate the dataset, $\boldsymbol{\alpha}$'s are required as well. In the LSSP model, the prior distributions for the $\boldsymbol{\alpha}$'s are white noise. However, we observe that nearby $\boldsymbol{\alpha}$'s have similar values. Therefore, we used a smoothing function for generating the $\boldsymbol{\alpha}$'s:

$$\alpha_r = w_{r1} * w_{r2} + w_{r3},$$

where as before $\boldsymbol{w_r}$ denotes the r^{th} design point. The resulting $\boldsymbol{\alpha}$'s are between -2 and 2, which is desired by their prior distribution. We set $\boldsymbol{\rho} = [0.2, 0.1, 0.3]$ and $\mu = 0$. The parameter n_{base} was chosen to be 60. **3dim** consists of 50 datasets, each containing two FVNs, one of size 60 and one of size 80.

Next, we estimated the parameters of the n^*LSSP model. The estimated link probabilities can be found in Figure 6.18. Although the variance is much higher than in the 1-dimensional case, it is surprising that that mean is still so close to the true probability. In order to get more accurate estimates, we suggest using a much higher number of networks and/or larger networks which we will show at the end of this section.

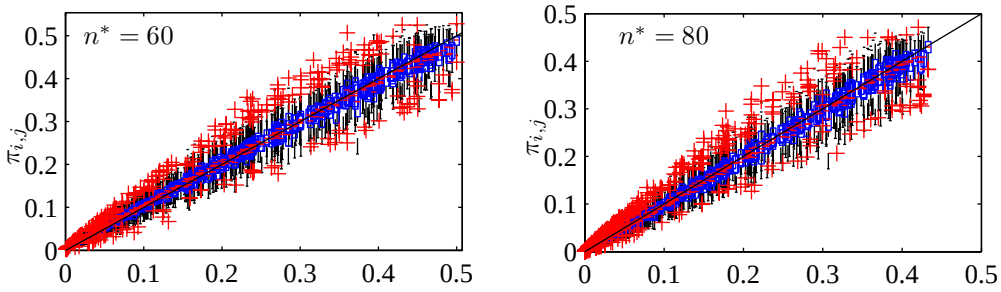


Figure 6.18: **3dim:** Boxplot of posterior mean probabilities.

In the second dataset (**5dim**), we calculated α_r as follows:

$$\alpha_r = 0.8[2(w_{r1} - 0.5) + w_{r2} + w_{r3} + 0.5(w_{r4} - 0.3) + w_{r5} - 2]$$

Furthermore, we set $\mu = 0$ and $\boldsymbol{\rho} = [0.16, 0.16, 0.16, 0.95, 0.5]$, *i.e.*, the first three dimensions have a high impact, whereas the 4th dimension has hardly any impact and the 5th dimension has

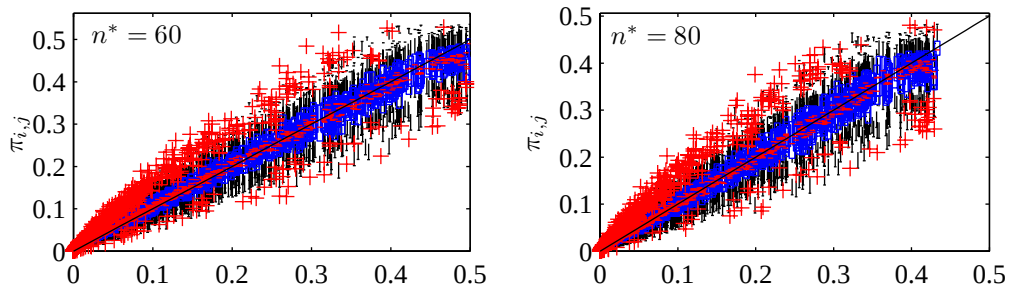


Figure 6.19: **5dim:** Boxplot of posterior mean probabilities.

mediocre impact. Here, we also wanted to show, whether or the n^*LSSP model is able to deal with dimensions which have a low impact (4th dimension). The boxplots for the posterior mean link probabilities can be found in Figure 6.19. Again, surprisingly despite only having two, very small networks as input, the mean estimates are pretty good. Again, for such high-dimensional cases, we suggest to provide much larger networks and possibly also increase the number of FVNs as shown based on an example next.

Increased FVN Sizes

We run experiments using the same parameter settings as in **3dim**, but increased the network sizes by a factor of 10, *i.e.*, the resulting networks had size 600 and 800. The estimated link probabilities were between 0 and 0.1589. Interestingly, the average absolute distance between the estimated probabilities and the true probabilities resulted in 0.0023 which we deem as a very good result. This shows that an increased network size can indeed improve the quality of the estimation substantially.

6.6 Parameter Sensitivity Analysis

In this section, we discuss the impact of the various model parameters.

Impact of μ

Based on our simulations, we find that the parameter μ is the parameter with highest impact on the link probabilities. In general, the larger μ, the larger the link probabilities. Also, μ determines the upper bound for the link probabilities.

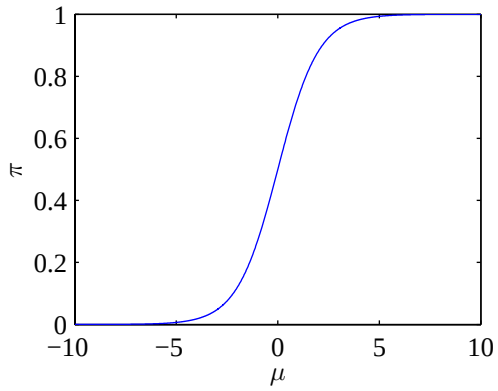


Figure 6.20: Relationship between μ and π.
Social space is ignored.

In Figure 6.20, the relationship between μ and π is depicted ignoring any impact of the covariates. Recall that

$$\pi = ilogit(\eta) = \frac{exp(\eta)}{1 + exp(\eta)}.$$

In Table 6.3 some interesting values for π and its corresponding μ values are listed. A link probability of $\pi = 0.1$ results in $\mu = -2.2$. If over 90% of the people should be connected, μ is 2.2. Also, μ values below -4.6 result in an almost empty network (the probability of an edge is 0.01) and μ values above 4.6 result in an edge probability of 0.99. The logistic function is symmetric around 0, *i.e.*, if around 50% of the people are supposed to be friends with each other, μ equals to 0. The μ value only determines the upper bound for the link probabilities. Therefore, the expected average degree in a network cannot be estimated only based on the parameter μ, but the impact of the social space needs to be taken into account. This social space is dependent on the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$ as discussed next.

Impact of $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$

The relationship between the social space and $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$ is very complex, see equation 6.1 and 6.2. Therefore, we restrict ourselves to some interesting cases. The first interesting case is the one in which the social space has no impact, *i.e.*, $z(x_i) = z(x_j) \forall x_i, x_j$. This case is depicted in Figure 6.6 and is a result of $\boldsymbol{\rho} = \vec{1}$ and/or $\boldsymbol{\alpha} = \vec{0}$. In this case covariates are completely ignored.

π	μ
0.01	-4.6
0.1	-2.2
0.5	0
0.9	2.2
0.99	4.6

Table 6.3: Interesting cases of π and their corresponding μ values (ignoring the social space).

The next case of interest is the one in which the social space is the same as the covariate space, this means $|z(\boldsymbol{x_i}) - z(\boldsymbol{x_j})| = |\boldsymbol{x_i} - \boldsymbol{x_j}|$. The latent function is able to model this case very well. The parameter estimates for a 100×100 dimensional sociomatrix and equi-distant covariates are:

- $\boldsymbol{\alpha} = [-1.26, -0.97, -0.62, -0.24, -0.15, 0.28, 0.80, 0.63, 0.98, 1.12]$
- $\rho = 0.53$

This is equivalent to replacing the function z by the identity function.

Interestingly, if $\boldsymbol{\alpha}$ consists of only similar positive, the resulting social space has a u-shape. For example, in Figure 6.21, the LSSP scores for $\boldsymbol{\alpha} = \vec{1}$ and two-dimensional covariates are shown. The impact of multiplying $\boldsymbol{\alpha}$ with a constant, can be seen in Figure 6.22(a), where $\boldsymbol{\alpha}$ is multiplied with 2, 3, 4, 5 and 6. The original graph is depicted in blue. Note that any distances in social space larger than 4.6 result in a link probability of almost 0. However, the impact of the distances in social space is dependent on the μ value, see Figure 6.23. On the x-axis the distance in social space can be found. On the y-axis, the impact on the link probability for various μ's is shown. For example for $\mu = 1$, a distance in space of 0.5 has a much higher impact than for $\mu = 2$ or $\mu = -2$.

In a multi-dimensional case, the impact of $\boldsymbol{\alpha}$ is very complex. Experiments of real-world data showed that close-by α_r's have similar values. The role of $\boldsymbol{\rho}$ is a very different. In Linkletter [48], it was shown, how the value of $\boldsymbol{\rho}$ can be used for variable selection. Recall that $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_d)$ is a d-dimensional vector. A ρ_l value close to 1 implies that the l^{th} dimension of the covariate has little impact on the response. This can be also seen in Figure 6.22(b), where the purple curve reflects the impact of $\rho = 1$. A low value of ρ_l reflects a high impact of dimension l of the covariates.

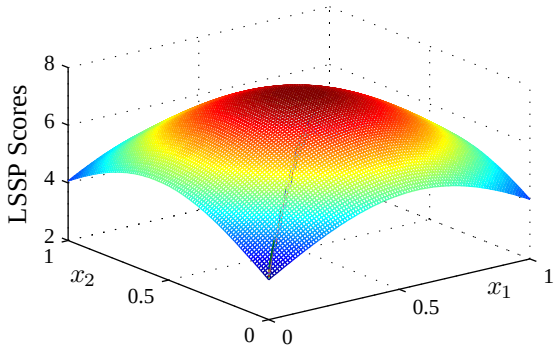
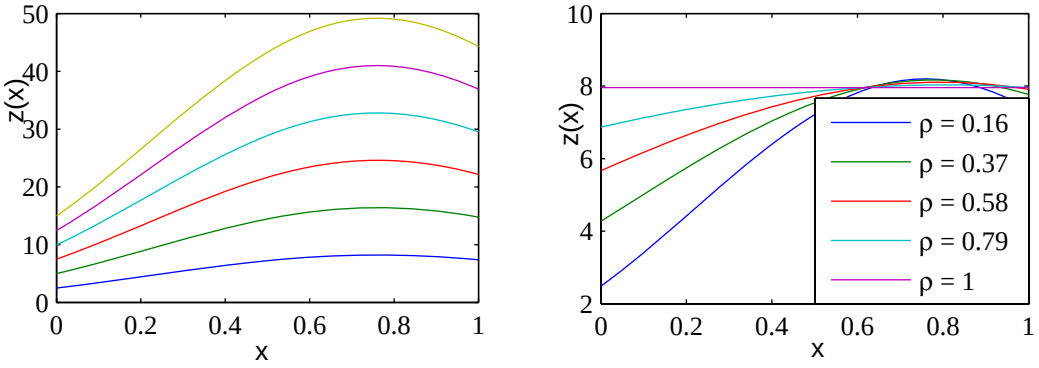


Figure 6.21: LSSP scores for $\boldsymbol{\alpha} = \vec{1}$ and $\boldsymbol{\rho} = 0.15$.



(a) LSSP scores for various $\boldsymbol{\alpha}$ values. Blue curve: $\boldsymbol{\alpha}$ values are the same as in Table 6.1; for the rest $\boldsymbol{\alpha}$ is multiplied by 2, 3, 4, 5 and 6.

(b) LSSP scores for various ρ values; $\boldsymbol{\alpha}$ is taken from Table 6.1.

Figure 6.22: Impact of various $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$ on LSSP scores.

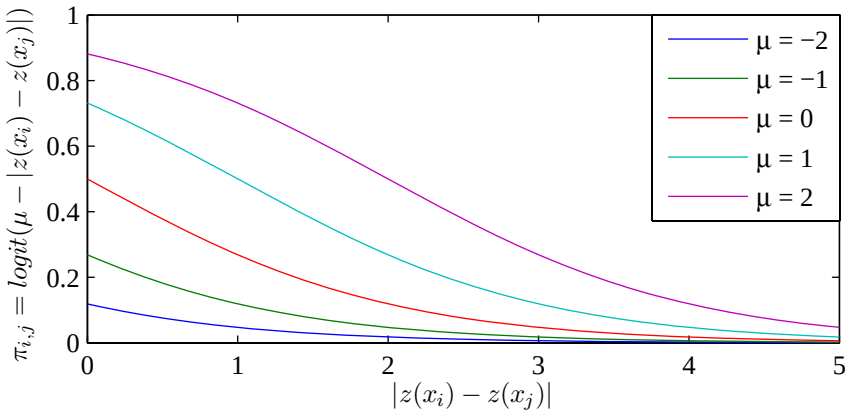


Figure 6.23: Impact of the distance in social space on the link probabilities for various μ's.

Impact of n_{base}, l and k

The parameter n_{base} should be set to the size of the smallest conceivable network. We already provided some intuition for the parameters l and k in Section 6.4.1 and 6.4.2. Recall that l impacts the overall link probability, whereas k impacts the distance in social space. Both parameters are between 0 and 1. Setting them to 0, yields the LSSP model. Setting l to 1, guarantees a constant average degree - given that the distribution of the covariates is the same (and independent of the samples size).

Impact of the covariates

The selection of covariates is very critical for simulating the desired network. Note that for the model the covariates have to be normalized to be between 0 and 1. In case the covariates are desired to be similar to the ones of the original network, we suggest to model the covariates by a Gaussian mixture model and draw the desired number of samples from this distribution. If the user wants to include communities in the social network, we suggest sample the covariates in a way, that they form clusters in social space. Such a cluster refers to an interval/area (or a set of intervals/areas) with similar LSSP values. For example, in Figure 6.3, grade 10 and 12 students are in the same cluster, since the have a very similar LSSP score.

How to determine the expected average degree in a simulated FVN

The expected degree in a network of size n^* is $(\sum_{i,j} Y_{i,j})/n^*$. The expected degree of a simulated FVN, $\mathcal{G}$ can be determined using the following formula:

$$\begin{aligned}\frac{1}{n^*}\sum_{i,j} Pr(Y_{ij}=1|\boldsymbol{x},\mu,\boldsymbol{\alpha},\boldsymbol{\rho},l,k,n_{base}) &= \frac{1}{n^*}\sum_{i,j}\pi_{i,j}^{n^*} \\ &= \frac{1}{n^*}\sum_{i,j}\left(\frac{n_{base}}{n^*}\right)^l p_{i,j}^{n^*} = \frac{(n_{base})^l}{(n^*)^{l+1}}\sum_{i,j} p_{i,j}^{n^*} \\ &= \frac{(n_{base})^l}{(n^*)^{l+1}}\sum_{i,j} ilogit\left(\mu - \left(\frac{n^*}{n_{base}}\right)^k |z(\boldsymbol{x_i}) - z(\boldsymbol{x_j})|\right)\end{aligned}$$

To calculate the average degree based on this formula is still very complex. However, by discretizing the covariates, a fast approximation can be achieved.

6.7 Conclusion and Future Work

In this chapter, we introduced a new methodology for simulating FVNs. This methodology is based on the LSSP model [47] and the n^*LSSP model. The n^*LSSP model was introduced in this chapter to accommodate effects related to the network size using two dampening functions. These functions regulate the link probability and the distance in social space. They also aid in controlling the expected degree in the simulated networks. Our experiments on simulated data show that the n^*LSSP model parameter can be estimated very well, even for only two very small networks.

There are several possible directions for future work. First, it might be worthwhile to investigate into an extension of the n^*LSSP model which integrates a cluster effect directly into the latent function, *e.g.*, by including a latent indicator function. Second, the μ parameter is constant and not dependent on the covariates. It would be interesting to see if μ can be made dependent on the covariates and at the same time avoid overfitting. In many networks, there are a few people with a very high number of friends. These people have an important role in the network. They are called *connectors*. A further extension of n^*LSSP model could include such people. It would be also interesting to see, if the n^*LSSP model can be used for privacy-preserving publishing of real social network data. In the current approach, links can be predicted for new people joining the network, if their covariates are known. When only the links of new people are known, the prediction of the covariates can be of great interest and would provide further insight into the model.

So far we assumed that all parameters in the n^*LSSP model can be estimated. However, in the absence of data, it might be interesting to provide recommendations for parameter values. In this context, we distinguish between two different scenarios: (i) a single FVN is provided as an input (ii) no FVN is provided as an input. In scenario (i), the parameters μ, $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$ can be estimated using the LSSP model. Based on the expected degree and other properties, it might be feasible to estimate the parameters l and k using MCMC methods. In scenario (ii), where no data is given, a user might have some intuition about certain network properties. It might be worthwhile to investigate which properties are necessary such that recommendations of parameter values are feasible. As a last direction for future research, we would like to extend the n^*LSSP model to deal with directed graphs.

Chapter 7

Conclusion

Feature vector networks (FVNs) are networks with node-specific feature vectors. An example for a FVN is an online social network, in which we know not only about the friendships between users, but also about their purchasing behavior. Data Mining for FVN is a fairly new research area with many real-world applications. However, there is a lack of publicly available datasets. Applications include community identification in social networks, module detection in Protein-Protein interaction networks and hotspot (areas with high criminal activities) detection. In this thesis, we introduced three research approaches in the area of Data Mining in FVNs which can be applied to the aforementioned problems: Connected X Clusters, Cohesive Pattern Mining and Simulation of FVNs. Future work of each of these approaches has been discussed in their respective chapters.

We introduced **Connected X Clusters**, a partitioning clustering algorithm for FVNs which is able to determine the number of clusters. Our algorithm outperforms state-of-the-art methods. The introduced clustering algorithm assumes clusters to be connected, but does not impose any requirement on their density. Furthermore, the feature vectors have to be similar in the full space, whereas in high-dimensional cases a subspace clustering approach might be more meaningful. Based on these observations, we designed an alternative clustering approach, Cohesive Pattern Mining, which is useful for applications in which density and subspaces are important.

In the chapter **Cohesive Pattern Mining**, we introduced the new concept of a cohesive pattern which combines dense subgraph mining with subspace clustering. Applications of this algorithm include identification of small communities and mining of modules in Protein-Protein interaction networks. We discussed several directions for future work, including adjusting our algorithm for weighted graphs or applying it to several FVNs/sets of FVNs. We also introduced a parallelized version of our algorithm.

One problem we faced, when working on the two previous research problems, was the lack of publicly available datasets. This motivated us to work on the **Simulation of Feature Vector Networks**. We introduced a new methodology for simulating FVNs. To the best of our knowledge, this is the first method which simulates FVNs such that feature vectors and network structure are dependent on each other. It is based on the previously introduced LSSP model and our extension, the n^*LSSP model. The n^*LSSP model takes into account effects related to the network size.

Summary

In this thesis, we introduced three approaches for Data Mining in FVNs. Our two proposed clustering/pattern mining approaches have been successfully applied to several research problems, in particular in the domain of social network analysis and computational biology, and show lots of potential for further applications. We also introduced the first simulation model for FVNs which promises to be useful in many research problems.

Bibliography

[1] Key facts about swine influenza (swine flu) spread of swine flu. In *Centers for Disease Control and Prevention. (http://www.cdc.gov/swineflu/key_facts.htm)*, 24 April 2009.

[2] R. Karp A. Ben-Dor, B. Chor and Z. Yakhini. Discovering local structure in gene expression data: The order-preserving submatrix problem. In *Journal of Computational Biology, 10(3-4)*, pages 373–384, 2003.

[3] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *International Conference on Management of Data*, 1993.

[4] R. Albert. Scale-free networks in cell biology. In *Journal of Cell Science 118*, 2005.

[5] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press, 1995.

[6] F. Bonchi and C. Lucchese. Pushing tougher constraints in frequent pattern mining. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2005.

[7] P.S. Bradley and U.M. Fayyad. Refining initial points for k-means clustering. In *International Conference on Machine Learning*, 1998.

[8] U. Brandes, M. Gaertler, and D. Wagner. Experiments on graph clustering algorithms. In *Algorithms - ESA 2003, 11th Annual European Symposium*, Budapest, Hungary, 2003.

[9] C. Carson, M. Thomas, S. Belongie, J.M. Hellerstein, and Jitendra Malik. Blobworld: A system for region-based image indexing and retrieval. In *Visualization*. Springer, 1999.

[10] A.-L. Cauchy. In *Cours d'analyse de l'École Royale Polytechnique, première partie, Analyse algébrique, Paris*, 1821.

[11] F.R.K. Chung and L. Lu. Spectral graph theory. In *CBMS Regional Conferenc eSeries in Mathematics*, 1997.

[12] R. Colak, F. Hormozdiari, F. Moser, A. Schonhuth, J. Holman, M. Ester, and S.C. Sahinalp. Dense graphlet statistics of protein interaction networks and random networks. In *14th Pacific Symposium on Biocomputing*, 2009.

[13] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 551–556, New York, NY, USA, 2004. ACM.

[14] C. Ding, X. He, H. Zha, M. Gu, and H. Simon. A min-max cut algorithm for graph partitioning and data clustering. *IEEE International Conference on Data Mining*, 2001.

[15] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[16] P. Erdós and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci.*, 5:17–61, 1960.

[17] M. Ester, R. Ge, B. J. Gao, Z. Hu, and B. Ben-Moshe. Joint cluster analysis of attribute data and relationship data: the connected k-center problem. In *SIAM International Conference on Data Mining*, 2006.

[18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1996.

[19] C. Stark et al. Biogrid: a general repository for interaction datasets. *Nucleic Acids Research*, (Database issue), 2006.

[20] Grioriev et al. A relationship between gene expression and protein interactions on the proteome scale: analysis of the bacteriophage t7 and the yeast saccharomyces cerevisiae. *Nucleic Acids Research*, 2001.

[21] R. Shyamsundar et al. A dna microarray survey of gene expression in normal human tissues. *Genome Biology*, 2005.

[22] T. Hughes et al. Functional discovery via a compendium of expression profiles. *Cell*, July 2000.

[23] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases, Fall 1996.

[24] U. Feige, D. Peleg, and G. Kortsarz. The dense k -subgraph problem. *Algorithmica*, 2001.

[25] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. New York: Wiley, 1968.

[26] P. Fjallstrom. Algorithms for graph partitioning: A survey. *Linkoping Electronic Articles in Computer and Information Science*, 1998.

[27] M. Garey and D. Johnson. *Computer and Intractability: a Guide to the theory of NP-Completeness*. 1979.

[28] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *Annual ACM symposium on Theory of computing*, 1974.

[29] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions and the bayesian restoration of images. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1984.

[30] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *International Conference on Very Large Data Bases*, 2005.

[31] M. Gladwell. *The Tipping Point: How Little Things Can Make a Big Difference*. Back Bay Books, January 2002.

[32] S. Guha, R. Rastogi, and K. Shim. Cure: an efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on Management of Data*, 1998.

[33] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *Information Systems*, 2000.

[34] L. Hagen and A.B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Trans. on Computed Aided Desgin*, 1992.

[35] M.S. Handcock, A.E. Raftery, and J.M. Tantrum. Model based clustering for social networks. In *Journal of the Royal Statistical Society, A, 170, 2*, pages 301–354, 2007.

[36] D. Hanisch, A. Zien, R. Zimmer, and T. Lengauer. Co-clustering of biological networks and gene expression data. *Bioinformatics*, 2002.

[37] R.A. Hanneman and M. Riddle. *Introduction to social network methods*. http://faculty.ucr.edu/∼hanneman/, 2005.

[38] P.D. Hoff, A.E. Raftery, and M.S. Handcock. Latent space approaches to social network analysis. In *Journal of the American Statistical Association*, volume 97, pages 1090–1098, 2002.

[39] R.S. Hunter. Photoelectric color-difference meter. In *Journal of the Optical Society of America, Vol. 38, Issue 7*, pages 651–651, July 1948.

[40] A. Jain and R. Dubes. *Algorithms for clustering data*. Prentice Hall, 1988.

[41] M.E. Johnson, L.M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. In *Journal of Statistical Planning and Inference 26*, pages 131–148, 1990.

[42] R.M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, 1972.

[43] G. Karypis, E.-H. Han, and V. Kumar. Chameleon: Hierarchical clustering using dynamic modeling. In *IEEE Computer*, 1999.

[44] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: Wiley, 1990.

[45] P.N. Krivitsky, M.S. Handcock, J.M. Tantrum, and A.E. Raftery. Representing degree distributions, clustering, and homophily in social networks with latent cluster random effect models. Technical report.

[46] P. Lazarsfeld and R. Merton. *Friendship as a social process: A substantive and methodological analysis*. Freedom and Control in Modern Society. Van Nostrand, 1954.

[47] C. Linkletter. *Spatial Process Models for Social Network Analysis*. PhD thesis, Simon Fraser University, 2007.

[48] C. Linkletter, D. Bingham, N. Hengartner, D. Higdon, and K.Q. Ye. Variable selection for gaussian process models in computer experiments. In *Technometrics, 48*, pages 478–490, 2006.

[49] H.J. Thiebaux M.A. and Pedder. *Spatial objective analysis with applications in atmospheric science*. London: Academic Press, 1987.

[50] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symposium on Mathematics, Statistics and Probability*, pages 281–297, 1967.

[51] P. McCullagh and J.A. Nelder. *Generalized linear models*. Monographs on Statistics and Applied Probability, Capman & Hall, London, 1983.

[52] M.D. McKay, R.J. Beckman, and W.J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. In *Technometrics (American Statistical Association)*, pages 239–245, 1979.

[53] M. McPherson, L. Smith-Lovin, and J.M. Cook. Birds of a feather: Homophily in social networks. In *American Review of Sociology 27*, pages 415–444, 2001.

[54] T. M. Mitchell. *Machine Learning*. New York; London: McGraw-Hill, 1997.

[55] F. Moser, R. Colak, A. Rafiey, and M. Ester. Mining cohesive patterns from graphs with feature vectors. In *SIAM International Conference on Data Mining*, 2009.

[56] F. Moser, R. Ge, and M. Ester. Joint cluster analysis of attribute and relationship data without a-priori specification of the number of clusters. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.

[57] M. Newman. The structure of scientific collaboration networks. In *Proc. Natl. Acad. Sci. 98*, 2001.

[58] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review*, 2004.

[59] M.E.J. Newman and E.A. Leicht. Mixture models and exploratory analysis in networks. *Proc. Natl. Acad. Sci. USA*, May 2007.

[60] A. Okabe, K.-I. Okunuki, and S. Shino. The sanet toolbox: New methods for network spatial analysis. In *Transactions in GIS*. Blackwell Publishing, July 2006.

[61] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: A review. *SIGKDD Explorations Newsletter*, 6(1):90, 2004.

[62] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2005.

[63] D. Pelleg and A. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Seventeenth International Conference on Machine Learning*, San Francisco, 2000.

[64] G. Salton and M. J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, 1983.

[65] J. Scott. *Social Network Analysis: A handbook*. Sage Publications, London, 2000.

[66] L.G. Shapiro and G.C. Stockman. New Jersey, Prentice-Hall, 2001.

[67] J. Shi and J. Malik. Normalized cuts and image segmentation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[68] J. Shi and J. Malik. Normalized cuts and image segmentation. In *Proc. of IEEE TPAMI*, 2000.

[69] M. Shiga, I. Takigawa, and K. Mamitsuka. A spectral clustering approach to optimally combining numberical vectors with a modular network. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.

[70] I. Ulitsky and R. Shamir. Identification of functional modules using network topology and high-throughput data. *BMC Systems Biology*, (8), 2005.

[71] S. Wasserman and K. Faust. *Social Network Analysis*. Cambridge University Press, 1994.

[72] S. Wasserman and P.E. Pattison. Logit models and logistic regressions for social networks: I. an introduction to markov random graphs and p. In *Psychometrika*, number 60, pages 401–425, 1986.

[73] Y. Wei and C. Cheng. Towards efficient hierarchical designs by ratio cut partitioning. In *IEEE Conf. on Computer-Aided Design*, 1989.

[74] S. White and P. Smyth. A spectral clustering approach to finding communities in graphs. In *SIAM International Conference on Data Mining*, 2005.

[75] S. Yu and J. Shi. Multiclass spectral clustering. In *International Conference on Computer Vision*, 2003.

[76] Z. Zeng, J. Wang, L. Zhou, and G. Karypis. Coherent closed quasi-clique discovery from large dense graph databases. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[77] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *ACM SIGMOD International Conference on Management of Data*, 1996.